UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

GENERACIÓN DE PATRONES DE PULSADO A PARTIR DE PISTAS MUSICALES

César Gómez López
Tutor: Carlos Aguirre Maeso

Mayo 2017

GENERACIÓN DE PATRONES DE PULSADO A PARTIR DE PISTAS MUSICALES

AUTOR: César Gómez López
TUTOR: Carlos Aguirre Maeso

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo de 2017

Resumen

Resumen Los **patrones rítmicos** son empleados en diferentes aplicaciones tales como videojuegos, aplicaciones multimedia y multimodales, microcontroladores de sistemas de iluminación, etc. Para implementar dichos patrones rítmicos se utilizan pistas pre-procesadas que con frecuencia incorporan el patrón rítmico implementado de forma manual. También suelen emplear pistas con sus diferentes canales de audios separados previamente, lo que facilita todo el proceso de análisis y generación de patrones. Algunas aplicaciones que hacen uso de este tipo de técnica son: Guitar Hero, Flash Dance, OSU, etc. Incluso podemos entrar en la puesta en escena de juegos de luces y láser de escenarios de conciertos o visualizadores de audio.

Este trabajo pretende generar dichos patrones rítmicos a partir de cualquier **pista de audio** general, es decir, sin canales de audio separados ni un procesado previo. De esta manera, se podrá emplear como **librería** para la creación de videojuegos rítmicos, así como escenarios de luces y láser o visualizadores de audio entre otras múltiples aplicaciones posibles, el límite estará en la imaginación del usuario.

Para conseguir este objetivo, se ha programado en lenguaje **C#** una **librería** que permite añadir diversos componentes de audio a objetos o elementos del entorno de desarrollo de videojuegos y aplicaciones multimedia **Unity**. Además, se han implementado diversos ejemplos sencillos de posibles aplicaciones de la **librería** creada, aunque se puede emplear para múltiples usos que usuario pueda imaginar. En esencia, la **librería** generará eventos, y el usuario decidirá qué tipo de eventos usar y qué hacer con ellos.

El resultado final del trabajo es una **librería** general y fácil de usar que permite generar los patrones rítmicos de cualquier audio deseado. Pese a que admite cualquier audio, los resultados serán más satisfactorios cuanto menos ruido y más limpia sea la pista musical. Además, la **librería** estará estructurada de manera que cualquiera la pueda modificar e investigar cambiando ciertos parámetros generales logrando resultados diferentes y aprendiendo más sobre el análisis del audio.

Palabras clave Patrones rítmicos, Unity, librería, pista de audio, C#.

Abstract (English)

Abstract The **rhythmic patterns** are used in different applications such as in videogames, multimedia and multimodal applications, microcontrollers of lighting systems, etc. To apply said **rhythmic patterns**, pre-processed tracks are used which frequently incorporate the **rhythmic pattern** carried out manually. Tracks with their different audio channels priory separated are also used allowing the entire process of analysis and creation of patterns to be made easier. Some applications that use such method are: Guitar Hero, Flash Dance, OSU, etc. In addition, it is possible to enter in the layout of light and laser displays of concert scenarios or audio browsers.

This project endeavors to create such **rhythmic patterns** from neither any general **audio tracks**, that is without separate audio channels nor any prior processing. In this way, it can be used as a **library** for the creation of rhythmic videogames, as well as laser and light scenarios or audio browsers between possible multiple applications, the only limits are those of the user's imagination.

In order to achieve this objective, the same has been programmed using the **C#** language, a **library** that allows adding several audio components to objects or elements of the videogame development settings and the **Unity** multimedia applications. Additionally, various possible simple examples have been implemented from the developed **library**, although it can be applied to multiple uses that the user can think up. In essence, the **library** will generate events and the user decides which events to and how to use them.

The result of the project would be a general **library** which would be easy to use and allowing to create any audio **rhythmic patterns**. Although it can admit any audio, better results can be obtained with reduced noise levels and clearer **music track**. Additionally, the **library** would be structured in such a manner that would make it possible to alter or inquire into by modifying some general patterns thus obtaining varying results and learning more on the audio analysis.

Key words Rhythmic patterns, Unity, library, audio track, C#.

Agradecimientos

Agradezco ante todo a mi madre, **Ana**, mi padre, **César**, mi hermana **Alicia,** y mis abuelas **Pili** y **Mamen** por haberme apoyado desde siempre, no habría llegado hasta aquí sin ninguno de ellos.

Gracias también a todos mis compañeros y amigos que han estado a mi lado todos estos años y me han hecho evolucionar como persona. En especial, gracias a **Celia** y **Jimena**, sin ellas, estos años en la universidad no hubiesen sido tan buenos. No puedo olvidarme tampoco de **Ibarra**, **Adrián** y **Adriana**.

A todas las personas que me han ayudado realizando las pruebas y valorando el proyecto, mis "conejillos de indias".

Por último, quiero ofrecer un agradecimiento especial a **Carlos Aguirre**, por aceptar mi proposición de proyecto y permitirme hacerla realidad.

Gracias a todos.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Este proyecto busca dar un soporte general y fácil de usar para incorporar patrones rítmicos de audio a diferentes aplicaciones tales como videojuegos, aplicaciones multimedia y multimodales, espacios virtuales, etc. El límite de las aplicaciones posibles vendrá dado por la propia imaginación del usuario de la librería.

Una de las ventajas de la librería es la gran libertad que aporta a futuros usuarios para crear videojuegos musicales, escenarios de luces, etc.
Al unir la temática de sonido a la de videojuegos, dos campos que atraen a una gran mayoría de gente, se consigue un trabajo que permitirá a multitud de usuarios que desconocen las características del sonido digital poder crear infinidad de escenarios posibles que precisen del uso de patrones rítmicos de una manera relativamente sencilla e intuitiva.

Nos encontramos en un momento en el que los videojuegos y la música forman parte de la vida diaria de gran parte de la población, y mucha gente empieza a investigar y desarrollar pequeños videojuegos y aplicaciones por su cuenta, ya sea por el atractivo que tiene poder realizar tu propio juego y ver que el resto lo disfruta, por simple curiosidad, etc. Para estas personas, el poder contar con librerías que les permitan hacer realidad sus ideas, supone una gran ayuda y apoyo para conseguirlo.

Unity es un entorno de desarrollo multiplataforma disponible para Microsoft Windows, Linux y OS X que tiene a su vez soporte de compilación de aplicaciones para otros tipos de plataformas tales como Android, Smart TV o Consolas de Nintendo, Sony o Microsoft. Actualmente es de los motores más utilizados por los usuarios que se inician en el mundo del desarrollo de videojuegos y es considerado un estándar de facto. Por ello esta librería, y sus diversos ejemplos de uso, están desarrollados para Unity.

Por otro lado, la realidad virtual cada vez está más presente en este mundo, y Unity permite crear escenarios en 3D para Oculus Rift, pudiendo crear espacios tridimensionales de luces que reaccionen a una pista musical de audio a partir de la librería implementada.
Como vemos, se podrá emplear para poder hacer realidad una inmensa cantidad de ideas que puedan estar en la mente del usuario de la librería.

1.2 Objetivos

El objetivo principal de este proyecto consiste implementar una librería que combine las diferentes teorías relacionadas con el análisis de audio con la generación de patrones rítmicos a partir de pistas musicales usando el entorno de desarrollo y soporte de Unity.

El usuario deberá ser capaz de usar de manera sencilla e intuitiva la librería desarrollada para poder emplear de una manera libre y creativa los patrones rítmicos que se detectan con el análisis de pistas de audio, sin depender de la necesidad de audios pre-procesados o especiales.

Para ello se definen pequeños objetivos que se deben ir implementando para conseguir la meta final: **Separar los componentes del proyecto, preparar las clases básicas necesarias, implementar las clases principales.**

1.2.1 Separar los componentes del proyecto

Se puede dividir el proyecto en cinco bloques principales, los cuales se explicarán en el apartado de diseño:

- Análisis
- Beats
- Librería de FFTs
- Detector de cambios
- Gestor de eventos y herramienta principal

1.2.2 Preparar las clases básicas necesarias

Es necesario preparar las clases básicas que representen un beat, un pico, que guarde un análisis realizado y una clase de apoyo general, que tendrá funciones que usen el resto de componentes de manera general, lo que vendría a ser una clase de utilidades.

1.2.3 Implementar las clases principales

Una vez se disponga de todos los preparativos previos, podremos proceder al desarrollo principal del proyecto de manera organizada. De esta manera se implementarán las clases que analicen audios, busquen beats, detecte picos y cambios significativos en las pistas, preparen eventos y procesen los mismos.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Capítulo 1 – Introducción**

  Motivación y objetivos del proyecto.

- **Capítulo 2 – Estado del arte**

  Conceptos de sonido digital, relación entre patrones ritmos, videojuegos y espectáculos lumínicos, presentación del entorno Unity 3D.

- **Capítulo 3 – Diseño**

  Diseño de la librería desarrollada.

- **Capítulo 4 – Desarrollo**

  Desarrollo del proyecto con muestras de código.

- **Capítulo 5 – Integración, pruebas y resultados**

  Presentación de pruebas de caja negra, los test realizados y los resultados obtenidos.

- **Capítulo 6 – Conclusiones y trabajo futuro**

  Conclusiones finales y planificación de trabajo futuro.

2 Estado del arte

En este capítulo se contará brevemente las características del audio digital, así como un breve resumen de su historia y evolución. Además, se mostrarán diversas aplicaciones más relacionadas directamente con el proyecto.

2.1 El sonido digital

El sonido digital es la codificación digital de una señal eléctrica que representa una onda sonora. Consiste en una secuencia de valores enteros que se obtienen mediante dos procesos: el muestreo, que consiste en fijar la amplitud de la señal eléctrica a intervalos regulares de tiempo denominados tasa de muestreo, y la cuantificación de la señal eléctrica, que consiste en convertir el nivel de las muestras fijadas en el proceso anterior en un valor entero de rango finito y predeterminado.

2.2 Los videojuegos musicales

PaRappa the Rapper es considerado el primer videojuego musical según Guinness World Records. Es de origen japonés y se lanzó en 1996 para PlayStation. El jugador debía completar combinaciones rítmicas diversas para superar los niveles del juego. Consistía en repetir los patrones de botones que realizaba un personaje, los cuales encajaban con el ritmo de la música del juego.

A partir de este primer *PaRappa the Rapper* o, con los años han ido apareciendo multitud de diversos estilos de videojuegos musicales.

2.2.1 Simuladores de baile

Videojuegos como Flash Dance o Dance Dance Revolution supusieron una revolución como simuladores de baile, apareciendo inicialmente en salones de videojuegos debido al gran espacio que necesitaban. Los jugadores deberían situarse sobre una pista o plataforma de baile con cuatro flechas como botones, las cuales debían pisar siguiendo el ritmo de la música y el patrón indicado en la pantalla.

Figura 2.1 Simulador de Dance Dance Revolution

Este tipo de juegos también se portaron a las videoconsolas domésticas, en las cuales, en ocasiones contabas con una alfombrilla que simulaba la pista de baile, o simplemente podías seguir el patrón de flechas con el mando de la videoconsola.

2.2.2 Simuladores de instrumentos

En este campo, lo primero que nos puede venir a la cabeza es el ya icónico y extendido videojuego *Guitar Hero*, lanzado por primera vez en noviembre de 2005.

En *Guitar Hero* se pretende simular que se toca una guitarra, para lo cual, al igual que en la inmensa mayoría de los juegos musicales, se debe pulsar los trastes simulados de la guitarra al mismo tiempo que aparecen en la pantalla. Además, se cuenta con una barra de rasgueo y una de trémolo, ofreciendo una simulación bastante completa de una guitarra eléctrica. Podríamos considerar como una versión de software libre, muy similar a *Guitar Hero*, el videojuego *Frets On Fire*. Además, otros simuladores, como Rocksmith, permitían la conexión con una auténtica guitarra eléctrica.

Figura 2.2 Ejemplo de Guitar Hero Live

Pero no solo existen videojuegos que simulan guitarras, también podemos encontrarnos con simuladores de múltiples instrumentos, poniendo de ejemplo los simuladores de baterías, como *Rock Band*, o incluso de una mesa de mezclas como *DJ Hero*.

2.2.3 Otros

Dejando a un lado los simuladores, también podemos encontrarnos con multitud de videojuegos musicales que toman como base los patrones rítmicos de pistas musicales. Podemos entrar en géneros de juegos de plataformas, poniendo de ejemplo *Electronic Super Joy Boy* o *Geometri Dash*, géneros de minijuegos como *Rythm Paradise*, u otros juegos extremadamente populares y extendidos como *Osu!*. En este último, además, juega gran importancia la comunidad de jugadores, ya que todos los mapas de juego son creados por los propios usuarios del mismo.

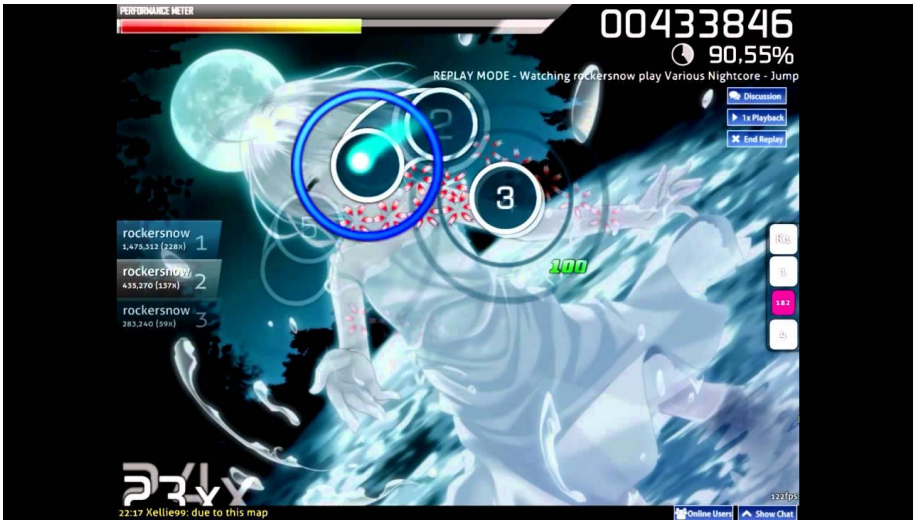

Figura 2.3 Pantalla de juego de Osu!

2.3 La música y la luz

La luz es un elemento que cada vez ha ido siendo más usado en relación con la música en espectáculos y conciertos. Es un elemento que se puede usar libremente a partir de la imaginación de las personas, creando escenarios que impactan a la gran mayoría de la gente. Un aspecto importante de esta relación música-luz recae en que normalmente dichos espectáculos de luz suelen seguir el ritmo de la música a la que acompañan, logrando una armonía atractiva audiovisualmente. Para ello se emplean microcontroladores que permiten controlar la posición, orientación o intensidad de dispositivos luminosos como cañones de luz, emisores de láser, espejos, etc., utilizados en la iluminación y animación de eventos (salas de fiesta, conciertos, festivales, discotecas, mítines, etc.).

Figura 2.4 ***Uso de luces en el festival de música electrónica Tomorrowland***

2.4 Unity 3D

Unity 3D es una herramienta de diseño e implementación de videojuegos que ha revolucionado el mercado. Una de sus principales características es la capacidad de generar versiones para distintas plataformas como pc, videoconsolas o contenido online. Esto es gracias a que utiliza C# y JavaScript bajo el intérprete Mono Runtime. No sólo tiene entorno de desarrollo para Windows, sino que también para Linux. Proporciona una perfecta capa de abstracción sobre la plataforma final de ejecución de la aplicación, de modo que no es necesario generar distintos proyectos para realizar un desarrollo de aplicación multiplataforma. Un mismo proyecto puede ser exportado a cualquier plataforma soportada por Unity, o incluso a varias a la vez. Genera aplicaciones para Android, Smart TV y consolas de Nintendo, Sony y Microsoft, además de para PC.

Otra de las características clave de Unity es que incluye herramientas para diseñar en 3D, generar animaciones, así como un potente generador de terrenos y un motor de físicas. Antes de Unity 3D, estas características se tenían que desarrollar por separado con programas como Blender, el software de modelado 3D y animación de código abierto. Además, proporciona una tienda online donde se puede comprar u obtener gratis determinados componentes para incluirlos en nuestros juegos y aplicaciones.

Una de las desventajas con las que cuenta este software es que su curva logarítmica de aprendizaje describe un avance lento; expresado en otras palabras, el proceso de su aprendizaje se prolonga en el tiempo debido a que la cantidad de conceptos y tecnologías a adquirir puede llegar a ser muy costoso. Aun así, los desarrolladores cuentan con una colección amplia de tutoriales en los que se esfuerzan en ayudar de manera gratuita a la comunidad. Además, el tiempo de aprendizaje invertido se compensa con la gran calidad

de los resultados que se pueden obtener a través de Unity 3D. Existe también una gran cantidad de foros y sitios web donde la comunidad resuelve dudas. En España destaca unityspain [5].

En conclusión, Unity 3D ha llegado a ser una de las mejores tecnologías para el desarrollo de videojuegos adoptando una filosofía que premia a la comunidad y da facilidades para aprender. También ofrece una licencia personal gratuita que permite el desarrollo comercial, siempre y cuando la capacidad de ingresos o los fondos recaudados no superen los 100.000$ por ejercicio fiscal. Aunque no es un software cerrado, como toda empresa busca un beneficio económico. Dicho beneficio lo consigue gracias al soporte técnico y al desbloqueo de herramientas, útiles en su mayoría cuando uno se embarca en un proyecto verdaderamente grande en Unity 3D, con lo que no afecta al desarrollador que solo quiere aprender o hacer algo simple. Por este motivo, además de por ser el estándar empleado en la EPS para las clases de videojuegos en Master y Grado, se ha elegido como plataforma base para el desarrollo de este TFG.

3 Diseño

Para realizar el proyecto se vio conveniente, para una correcta estructura y organización, separar el mismo en diferentes partes

- Análisis
- Beats
- Librería de FFTs
- Detector
- Gestor de eventos y herramienta principal

3.1 Análisis

Se encargará de analizar un audio a partir de su magnitud de espectro. Contaremos con una clase que analice el audio y otra que almacene los datos recopilados.

3.1.1 Analizador

A partir de un audio cualquiera, deberá detectar y guardar para cada momento o frame del mismo los picos u onsets (inicios de notas) de la pista, así como otros datos relevantes como la magnitud del sonido o el flujo entre los diferentes frames.

3.1.2 Datos del análisis

Los datos que se recopilen desde la clase principal de análisis serán guardados en una clase básica para poder acceder a los mismos de manera sencilla.

3.2 Beats

También conocido como latido o pulsación del audio, se encargan de llevar el ritmo base y, normalmente, constante del audio.

3.2.1 Beat

Guardará los datos correspondientes a un Beat detectado.

3.2.2 Buscador de beats

Esta clase será la encargada de "trackear" o buscar los beats de una pista de audio, los guardará y además calculará los beats por minuto (bpm) del sonido.

3.3 Librería de FFTs

Esta librería no será implementada, sino que se empleará una ya existente [4]. Se trata de un conjunto de funciones para operar con Transformadas de Fourier, las cuales son un elemento clave en el análisis de frecuencias selectivas, es decir, se podrá adaptar el análisis a cada audio según las diferentes frecuencias del mismo, pudiéndose precisar más en la detección de picos y beats.

3.4 Detector de cambios

Será necesaria una clase que se encargue de detectar los cambios a partir de los datos guardados tras el análisis de la pista de audio. De esta manera, se podrá actuar de una manera u otra según haya aumentos o descensos en la intensidad del audio, en el flujo de frames, etc.

3.5 Gestor de eventos y herramienta principal

Por último, serán necesarias dos clases principales que se encarguen de preparar eventos según los datos y cambios recogidos por los anteriores bloques, y se realicen acciones en consecuencia de los mismos.

3.5.1 Eventos

Se implementará una clase preparadora de eventos de beats, picos, cambios de frames, y gestor de tiempos según los datos analizados y procesados por el proyecto. Esta clase será uno de los dos componentes de Unity del proyecto, a partir del cual el usuario podrá seleccionar los datos del audio que utilizará y el método de la clase mediante el cual los recogerá.

3.5.2 Herramienta principal

Esta clase se encargará de recoger los eventos generados y gestionar las acciones a realizar necesarias según los diferentes tipos de eventos. También gestionará las funcionalidades del audio e inicializará los análisis y recogida de datos del mismo. Será el segundo componente de Unity en el proyecto y dará la opción de calcular a tiempo de ejecución los datos o de pre-analizar el audio antes de pasar a futuras aplicaciones con los datos del mismo.

3.6 Unity 3D

3.6.1 C#, JavaScript y Mono Runtime

Para el desarrollo de código se usan los lenguajes C# y JavaScript, que se interpretan en Mono Runtime. Debido a ello destaca su capacidad multiplataforma. No existe diferencia en las funcionalidades que aporta cada lenguaje a nivel de Unity, siendo a gusto del programador el uso de uno u otro. Aun así, es preferible el uso de JavScript para la creación de funcionalidades simples y de fácil diseño, dejando el resto a C#.

C# tiene varias ventajas, entre la que se encuentra que es un lenguaje tipado y no tipado al mismo tiempo. Es decir, tiene clases y tipos de datos comunes y básicos como enteros (int), arrays (string), decimales (double), etc., pero además posee un tipo genérico que permite una programación no tipada. Este tipo de dato se usa haciendo declaraciones a través de var, que se considera palabra reservada. C# es un lenguaje que se asemeja a Java desde el punto de vista de que se ejecuta sobre una máquina virtual que traduce a código máquina. La máquina virtual se llama Common Language Runtime(CLR) y admite otros lenguajes como F#, lo que permite crear proyectos mixtos. Unity 3D utiliza Mono Runtime, que es la versión de código abierto de CLR. Mono Runtime es un intérprete que se encarga del manejo de las llamadas a sistema operativo y de la ejecución del código. Controla y gestiona la memoria de los programas y su liberación a través del Garbage Collector. C# es un lenguaje con orientación a objetos que incluye la característica de la Herencia Múltiple, que permite que una clase obtenga propiedades de más de un tipo distinto de clase padre.

Dado que utiliza Mono Runtime, se puede usar el IDE Mono para programar, siendo este el principal en OS X, aunque se potencia más el uso de Visual Studio, el IDE principal de Windows. Uno de los defectos de Visual Studio es su tamaño, debido a que incorpora herramientas para desarrollar con el framework Windows Forms y Xamarin Forms. Pese a ello, al utilizar Windows para la realización del TFG, he decidido usar Visual Studio para la codificación de los scripts que han sido necesarios.

3.6.2 Descripción de Unity 3D

3.6.2.1 Elementos Gráficos: GameObjects y escenas

La clase GameObject es la clase base de todo elemento gráfico de Unity 3D. Incluye funciones para instanciarlo, destruirlo, generar transformaciones sobre él u obtener referencia sobre GameObjects similares. También puede contener en su interior otros GameObjects. Un elemento 2D que represente un botón o un panel sobre el que dibujar también es un GameObject. [10]

Se instancian en los elementos denominados escenas. Una escena es una representación tridimensional en primer lugar del espacio donde se va a representar el juego. Para juegos en 2D también se usan escenas solo que se juega con las cámaras para que no tenga efecto 3D.

Todo conjunto de GameObjects puede guardase en una plantilla para poderse reutilizar más adelante. Dichas plantillas se llaman Prefabs y están creadas para realizar un uso y diseño modular de todos los objetos que creemos. Para incorporarlo al juego solo tenemos que pinchar en él en la ventana de assets y arrastrarlo a la posición que deseemos dentro de la escena.

3.6.2.2 Clase base de control: MonoBehaviour [11]

La clase MonoBehaviour proporciona acceso al ciclo de vida que tiene un script (código a ejecutar en una escena) dentro de un GameObject. Similar al ciclo de vida de una aplicación de Android. Proporciona un método para inicializar variables y atributos, para liberar recursos, y para ejecutar acciones en cada frame (unidad de refresco de imagen) del juego. También proporciona acceso a funcionalidades de Unity 3D como la instanciación de GameObjects.

El orden de ejecución, de primero a último, es el que se muestra a continuación. De cada apartado se muestran los más relevantes, pues puede llegar a haber varias rutinas por apartado:

- **Editor**

  Reset Es el método que se invoca cuando el código es añadido al GameObject.

- **Cuando carga la primera escena**

  OnLevelWasLoaded Se invoca cuando la nueva escena o nivel ha sido cargado.

- **Antes de la actualización del primer frame**

  Start Es llamado antes de la primera actualización de frame solo si la instancia del script está activada.

- **Entre frames**

  OnApplicationPause Es el método que se invoca cuando se pausa la escena

- **Orden de actualización**

  FixedUpdate Es el método que se invoca cuando los frames por segundo (FPS) son demasiado bajos.

  Update Es el método que se invoca una vez por frame. En esta función se hacen cálculos de movimientos o lógica básica de movimiento.

  LateUpdate Es el método que se invoca justo después de Update. Se suele usar en cámaras de tercera persona o similar.

- **Renderizado**

  OnGUI Es el método que se invoca cuando ocurre un evento en interfaz gráfica. Dicho evento no tiene porqué ser siempre de tipo input.

- **Corrutinas**

  yield Se ejecuta cuando no hay más objetos a los que invocar la función Update.

- **Cuando el objeto es destruido**

  OnDestroy Es el método que se invoca cuando el objeto va a ser destruido o eliminado de la escena.

- **Cuando se abandona la escena**

  OnApplicationQuit Es el método que se invoca cuando la aplicación se va a cerrar.

Estos son los más usados o representativos, pero existen muchos más.

3.6.2.3 Recursos y componentes: Asset Store [12]

Una de las principales ventajas de Unity 3D es la capacidad para aprovechar el trabajo de otros para poder usarlo en nuestro beneficio personal. Como ya he comentado, el uso de Prefabs permite reutilizar componentes ya creados anteriormente. También se puede llevar a otro punto: existe una tienda virtual, llamada Asset Store en la que la comunidad y empresas pueden regalar o vender sus Prefabs y componentes para que otros puedan usarlos. A estos Prefabs y componentes se les llama Assets y van desde terrenos y animaciones a modelos 3D. Además, es la plataforma desde la que los desarrolladores de Unity 3D comparten y distribuyen los Assets básicos a los que puede acceder cualquiera que use Unity 3D.

3.7 Diagrama de clases

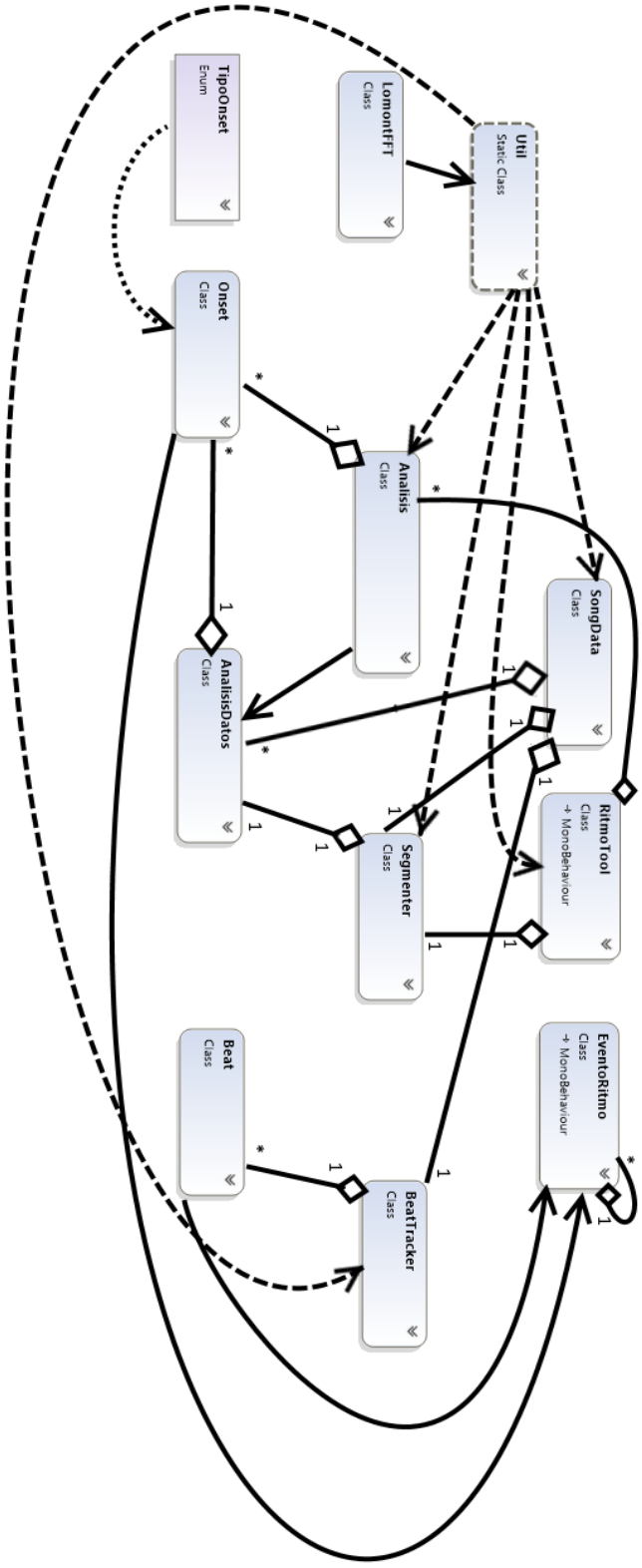

Figura 3.1 ***Diagrama de clases***

Con la finalidad de permitir que el diagrama sea claro y representativo, sólo se muestra el nombre de cada clase. A continuación, se podrá observar los datos de cada clase:

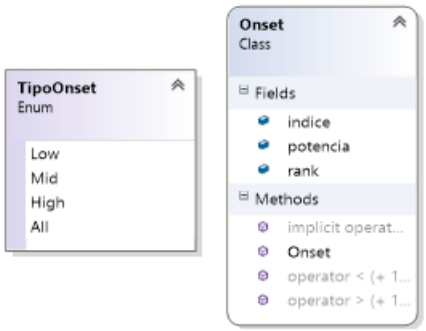

Figura 3.3 Clase Onset

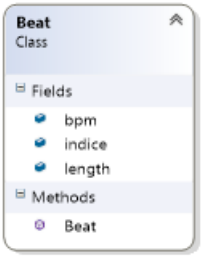

Figura 3.2 Clase Beat

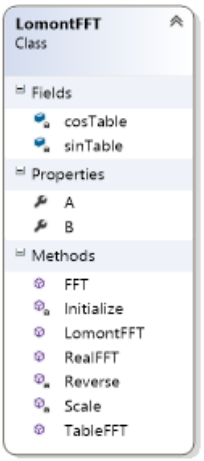

Figura 3.5 : Clase LomontFFT

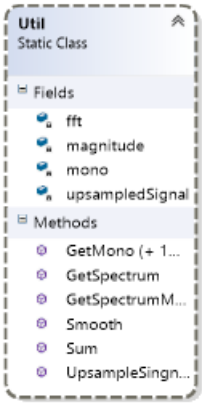

Figura 3.4 Clase Util

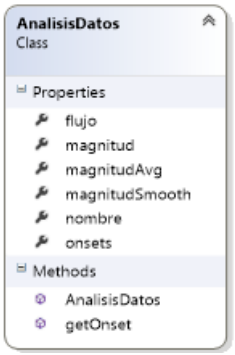

Figura 3.7 : Clase AnalisisDatos

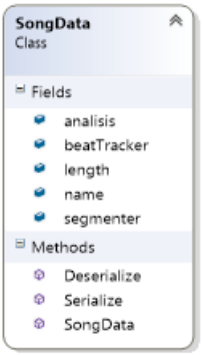

Figura 3.6 Clase SongData

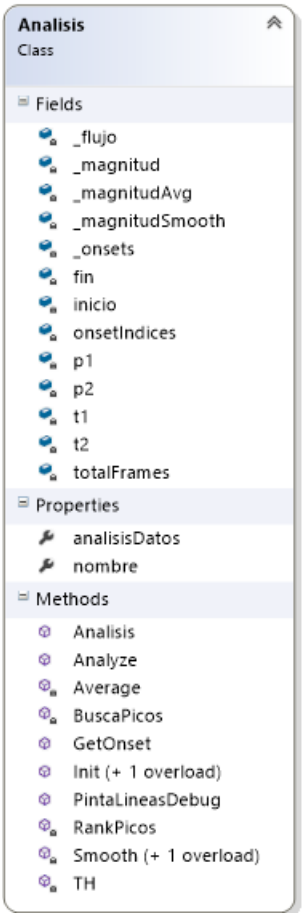

Figura 3.8 Clase Analisis

Figura 3.9 Clase BeatTracker

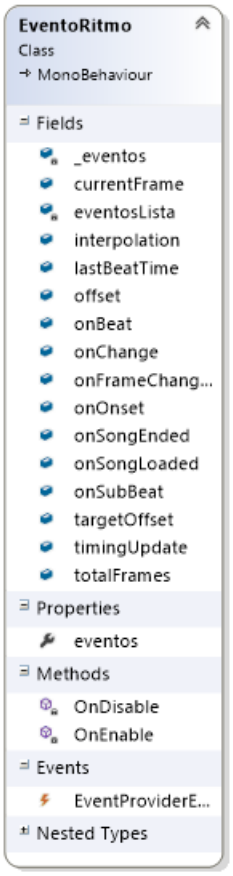

Figura 3.10 Clase EventoRitmo

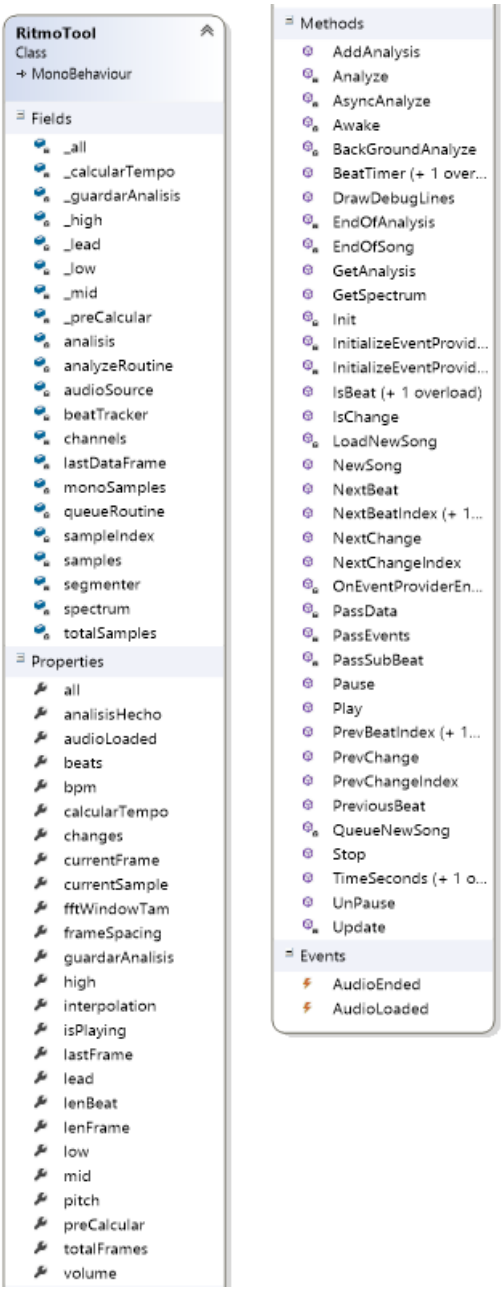

Figura 3.11 Clase RitmoTool

4 Desarrollo

En este apartado se comentará el uso de las funciones más relevantes de cada módulo de la librería.

4.1 Análisis

4.1.1 Analisis.cs

Esta clase servirá para analizar un audio a partir de su magnitud de espectro en diferentes momentos del mismo.

La clase tendrá en cuenta los onsets, las magnitudes y el flujo del audio, además de contar un objeto de tipo AnalisisDatos para guardar el análisis de la pista. Ver **Figura A.1.** En la **Figura A.2** y la **Figura A.3**, se ve como con las funciones Init podremos inicializar las variables de la clase a partir de un nuevo audio o de datos ya existentes.

La función mostrada en la **Figura A.4** se encarga de buscar picos relevantes en el espectro, lo cual nos permite encontrar las diferentes notas del audio. Así mismo, una vez detectados las notas a través de los picos del espectro, se les asigna un valor de ranking para diferenciar la intensidad de las mismas, lo cual permitirá en un futuro decidir con qué rangos de notas trabajar. Ver **Figura A.5**.

Las funciones Smooth, en la **Figura A.6,** permitirán tener una medida más estandarizada de las magnitudes recogidas, facilitando el uso de las mismas en un futuro.

Con Average, **Figura A.7**, se pretende tener medidas constantes, en cada momento de audio, con ello conseguimos que no haya continuas variaciones con cada instante analizado, sino que se calculará una media de los instantes de un momento dado y se les asignará el mismo valor consiguiendo unos resultados más fluidos y estables.

Por último, con una llamada a Analyze, función mostrada en la **Figura A.8**, podremos conseguir los datos más relevantes del espectro de audio para ser utilizados en un futuro por otras clases.

4.1.2 AnalisisDatos.cs

Esta clase será la encargada de almacena los datos obtenidos a partir de la clase Analisis.cs, pudiendo facilitar el acceso a los mismos. En la **Figura A.9** se observan los atributos guardados en la clase.

4.2 Beats

4.2.1 Beat.cs

La clase Beat.cs será la encargada de alamcenar los datos de un Beat. Podemos observarla en la **Figura A.10.**

4.2.2 BeatTracker.cs

BeatTracker se encargará de encontrar los beats del audio, así como de calcular los beats por minuto. Ver **Figura A.11.**

Al igual que con la clase Analisis.cs, contaremos con dos Init, según queramos inicializar el tracker a partir de datos ya existentes o a partir de un nuevo audio. Los podemos observar en la **Figura A.12** y la **Figura A.13**.

Para analizar en los frames consecutivos del audio y buscar un beat, se emplea la función TrackBeat, que aparece en la **Figura A.14**. Además, se pretende comparar una parte de la pista consigo misma para buscar repeticiones en el audio que serán tratadas más adelante. En la **Figura A.15**, se muestra FindBeat, que sirve para actualizar una repetición encontrada en el audio y calcular la longitud de beat y su offset más probable.

Hay dos tramos de una pista de audio que hay que tratar de manera especial, estos son el inicio y el final. Para ello se usarán las funciones FillInicio y FillFin. Los datos tomados con FillInicio servirán como referencia para los siguientes frames del audio, y en FillFin se tomarán los mejores últimos valores del mismo para rellenar la información de los beats encontrados. Ver **Figura A.16** (FillInicio) y **Figura A.17** (FillFin).

4.3 Detector de cambios

4.3.1 Segmenter.cs

La clase Segmenter nos servirá para detectar los cambios a partir de datos adquiridos como el flujo, la intensidad, etc. Podemos ver su constructor en la **Figura A.18**.

En la **Figura A.19** y la **Figura A.20**, observamos cómo volvemos a contar con dos inicializadores diferentes, según tengamos datos previamente guardados o un audio nuevo, al igual que ocurría con las clases de análisis y beatracker.

Para detectar cambios en el espectro, precisaremos de los datos recogidos tras un análisis, por ello si partimos de un nuevo audio, se deberá analizar antes de poder intervenir esta función. Se buscarán decrecimientos o incrementos de magnitud notables y se guardarán para un uso futuro por parte de otros módulos. Ver **Figura A.21**.

4.4 LomontFFT.cs

Esta clase implementa las evaluaciones de las FFT reales y complejas. No ha sido implementada, sino que ha sido recogida del grupo Lomont, el cual, además de esta librería posee otros algoritmos destacablesy útiles. El copyright pertenece a Chris Lomont, el cual permite usar su código libremente siempre que sea mencionado su procedencia. [4]

4.5 Util.cs

Esta clase da soporte al resto de módulos. Sus funciones son usadas en varios ficheros diferentes, por lo que era conveniente reunirlas en una clase separada del resto para más facilidad de uso y modularidad de código.

Entre ellas cabe destacar la función que permite obtener los datos del espectro de audio usando la FFT real partiendo directamente de los datos originales del audio que nos proporciona Unity. A partir de esta conversión podemos manejar los datos del espectro del audio para lograr nuestra finalidad. [2][3]

4.1 Componentes para Unity 3D

4.1.1 Generador de Ritmos

A través de la clase EventoRitmo.cs se generan los diferentes tipos de eventos: beat, subbeat, onset, change, timeUpdate, frameChanged y UnityEvent. Además de generar los eventos, contiene una lista que los guarda, convirtiéndose en su propia clase contenedora de eventos. A su vez, la clase se comporta como un componente de Unity 3D, el cual se puede conectar con los objetos de una escena. Esto permitirá conectar los eventos que se deseen usar a los scripts que se desarrollen para hacer uso de la librería, además de mostrar ciertos parámetros de debugeo, como pueden ser el frame actual del audio, los frames totales o la interpolación entre pulsos. También permitirá decidir el offset de pre-analizado en caso de que no se prefiera la opción de análisis y uso a tiempo real.

El uso de los diferentes tipos de eventos vendrá dado por el propio usuario de la librería. En las pruebas que se realizarán se podrá observar varios ejemplos de empleo de los mismos, mostrando la posibilidad de analizar y generar los eventos a tiempo real en paralelo a una reproducción de audio, o a través de un análisis previo a la reproducción.

En la **Figura A.22** podemos observar el código completo de esta clase generadora y auto-contenedora de eventos.

4.1.2 Herramienta de ritmo

Es la clase que gestiona la relación entre el audio importado y el resto de clases. Realiza las llamadas necesarias para el buen funcionamiento de la librería en conjunto, es decir, coordina los diferentes tipos de análisis, generación de eventos y el empleo de los mismos.

Como componente nos permite importar a la librería los audios a analizar, además de poder seleccionar la preferencia de pre-análisis de los mismos y guardado de los datos obtenidos. Al igual que el generador de ritmos, nos permite observar diferentes parámetros de debugeo: Frame actual, frames totales, último frame analizado, beats por minuto y longitud de beat.

En la **Figura A.23** podemos observar el código completo de esta clase.

5 Integración, pruebas y resultados

5.1 Pruebas de caja negra

Las pruebas de caja negra es un método de testeo en el que la estructura interna, el diseño y la implementación del programa no es visible o conocido para el realizador de las pruebas. Estos test pueden ser funcionales o no, pero normalmente sí lo son.

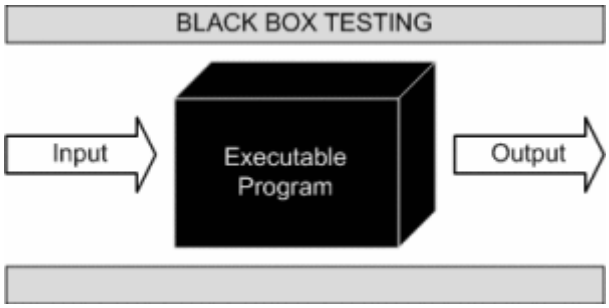

Figura 5.1 Test de caja negra

Este método es denominado de dicha manera porque el software del programa, para los ojos del encargado de realizar las pruebas, es como una caja negra, dentro de la cual no puede ver nada. Se usa con la intención de encontrar errores dentro de las siguientes categorías:

- Funciones incorrectas o perdidas.
- Errores de interfaz.
- Errores en estructuras de datos o accesos externos a bases de datos.
- Errores de comportamiento.
- Errores de inicio o terminación al ejecutar.

Por ejemplo, una persona sin conocimiento de la estructura interna del proyecto, puede probarlo usando el selector de audio (explicado más adelante) eligiendo un método de ejecución y un audio cualquiera para comprobar que funciona correctamente viendo las salidas visuales del programa y aportando información y opinión sobre el proceso.

Este tipo de test se pueden aplicar a nivel de test de integración, de test de sistema o de test de aceptación. Algunas de las técnicas que suelen ser usadas para la realización de estas pruebas son:

- **Partición equivalente**: Es un test de software que involucra la división de valores de entrada en particiones válidas e inválidas y selecciones representativas de los valores de cada partición como datos del test.
- **Análisis de valores de frontera**: Involucra la determinación los valores extremos para la entrada de datos, y selecciona aquellos que están en la frontera, además de aquellos que están dentro y fuera de la misma como datos del test.
- **Gráfico causa-efecto**: Esta técnica conlleva la identificación de los posibles casos o condiciones de entrada y los efectos o condiciones de salida, elaborando un gráfico causa-efecto y generando casos de test acorde al mismo.

5.1.1 Ventajas

Estas pruebas están realizadas desde el punto de vista del usuario y ayuda a encontrar dificultades o fallos en las especificaciones de uso. Además, no se necesitan conocimientos sobre lenguajes de programación o de cómo se ha implementado el software. Por ello pueden ser realizados por personas ajenas a la realización del proyecto, consiguiendo diferentes perspectivas que los desarrolladores pueden no haber tenido en cuenta, y pudiendo ser realizadas tan pronto las especificaciones estén completas.

5.1.2 Desventajas

Como consecuencia de este tipo de pruebas, solo una cantidad limitada de entradas posibles pueden ser comprobadas, por lo que muchos caminos del programa se quedarán sin probar. Al ser usuarios ajenos al desarrollo, sin unas especificaciones lo suficientemente claras, lo cual suele suceder en muchos proyectos, los casos de prueba serán difíciles de diseñar y probar, además de poder darse casos de redundancia de test si el diseñador o desarrollador del software ya ha comprobado dicho test.

5.2 Pruebas

Se han diseñado cuatro programas de prueba básicos distintos en los que se mostrarán diversas situaciones posibles en las que se puede usar la librería. Están serán:

1. Diseño de simulador de pulsado de guitarra.
2. Diseño de visualizador de audio general.
3. Diseño de visualizador de audio con intensidad.
4. Diseño de espectáculo de láser.

Los 4 tipos de pruebas generan los mismos eventos, pero son usados de diferentes maneras.

5.2.1 Diseño de simulador de pulsado de guitarra

Su principal función es comprobar el pre-análisis del audio con un offset de tiempo pre-establecido. Visualmente, esto quedará reflejado en que podremos ver como los pulsos rítmicos ya están preparados antes de que sucedan.

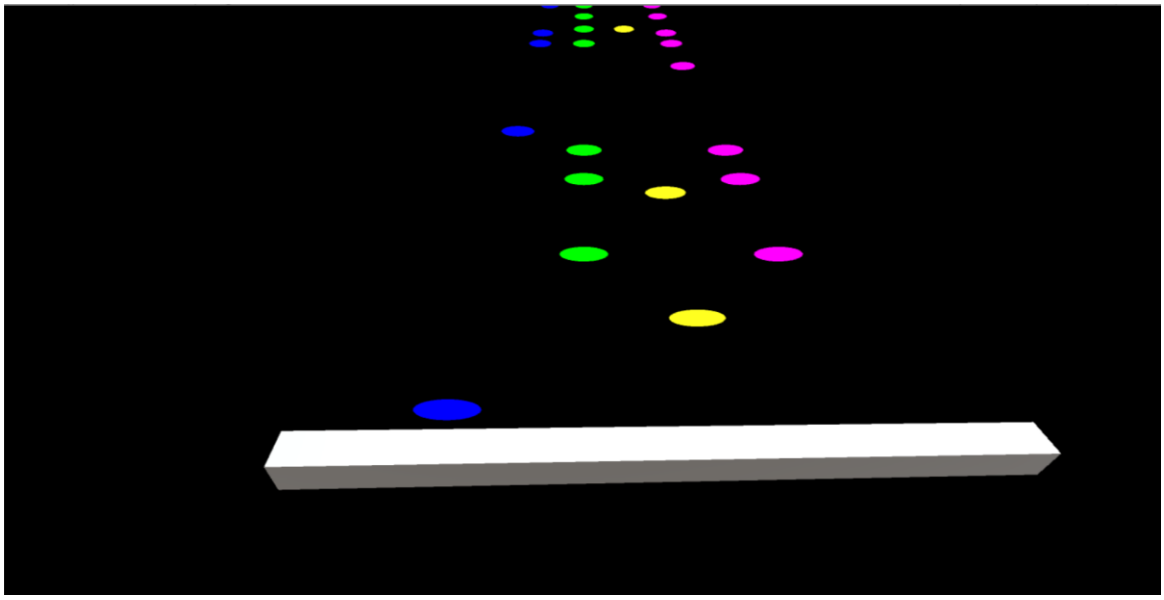

Figura 5.2 Simulador básico de pulsado de guitarra

Como observamos en la **Figura 5.2**, las notas de pulsado ya se muestran gracias a un análisis previo, adelantándose de esta manera al audio reproducido a tiempo real. Lospulsos descienden siguiendo el ritmo de la canción, de manera que podemos observar cómo las notas van acelerando o decelerando según el momento del audio.

5.2.2 Diseño de visualizador de audio general

Este programa de prueba simula un espacio 3D lleno de “estrellas” apagadas que se iluminan de un color u otro según la potencia de la nota detectada. En este caso el análisis se realiza a tiempo real, por lo que el resultado visual que se muestra se calcula al mismo tiempo en el que el audio se reproduce. Este ejemplo podría llevarse al terreno de la realidad virtual integrándolo con unas Occulus Rift [9].

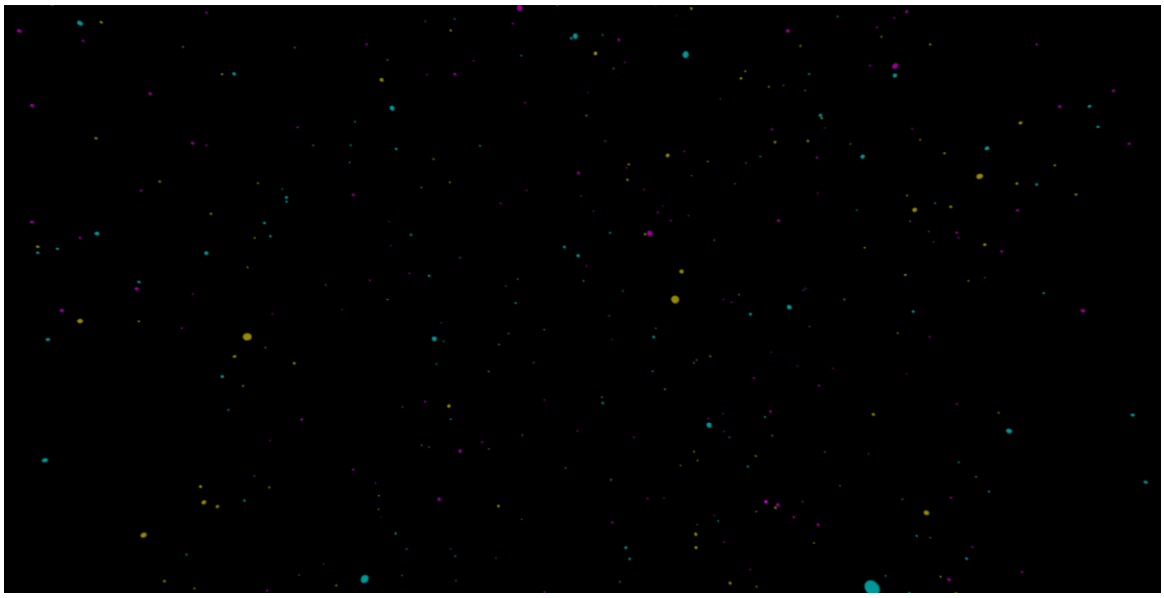

Figura 5.3 Visualizador básico de audio general

La **Figura 5.3** nos muestra la representación de un único momento del audio. A diferencia del programa de prueba del simulador de guitarra, en este ejemplo no podemos observar los pulsos futuros al momento de reproducción, ya que se generan y muestran al mismo tiempo.

5.2.3 Diseño de visualizador de audio con intensidad.

Este caso es similar al anterior visualizador, pero a diferencia de este, además de tener los diferentes tipos de notas ordenadas, podremos ver la intensidad de las mismas, de manera que una estrella brillara más o menos según la intensidad de la nota que representa.

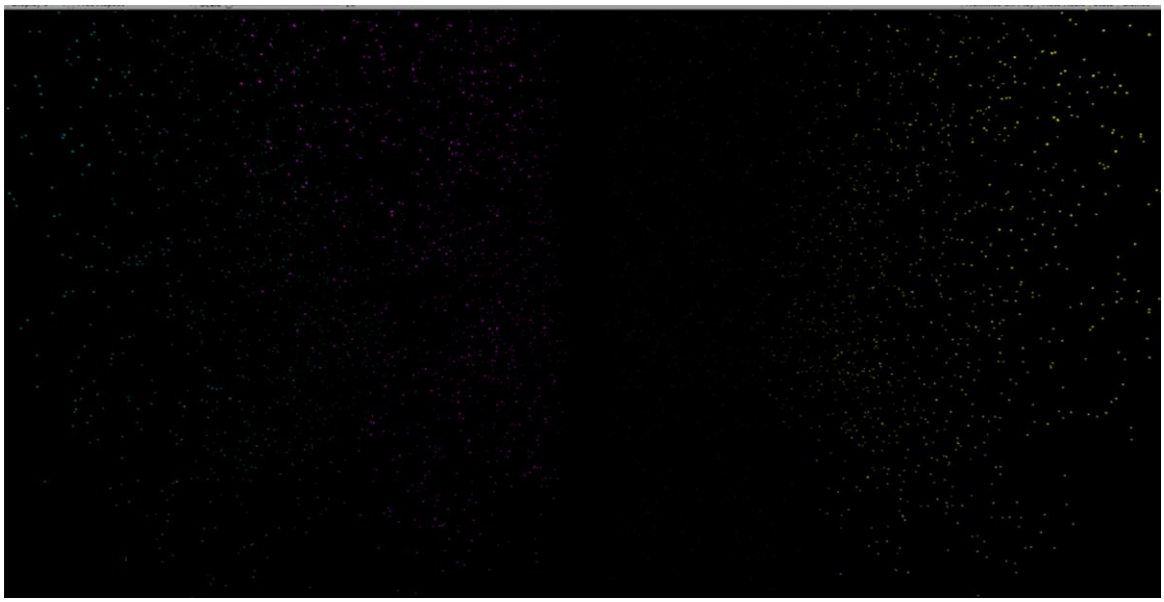

Figura 5.4 Visualizador básico con intensidad

Como se puede observar en la **Figura 5.4,** los distintos tonos de colores están más o menos iluminados en un determinado momento del audio, llegando al caso de que el tono verde de la tercera columna apenas puede apreciarse debido a su baja intensidad, mientras que los tonos amarillos y rosas se perciben perfectamente, y el tono azul un poco más apagado. De nuevo, este programa analiza a tiempo real de reproducción de audio.

5.2.4 Diseño de espectáculo de láser

Esta prueba tiene como finalidad aportar otro ejemplo más de uso de la librería. El análisis se realiza a tiempo real y busca mostrar una ejemplificación básica de uso de los eventos generados para generar haces de luz que podrían dar soporte a la organización de un espectáculo audio visual simplificando mucho el trabajo de la programación de las luces.

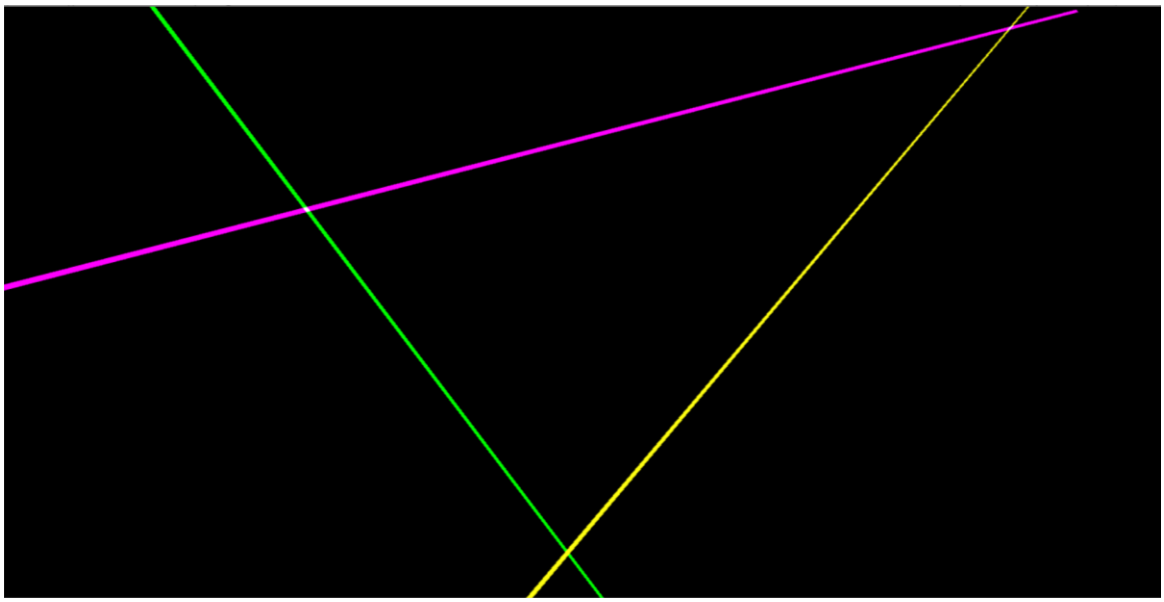

Figura 5.5 Planificador básico para espectáculo de luces 1

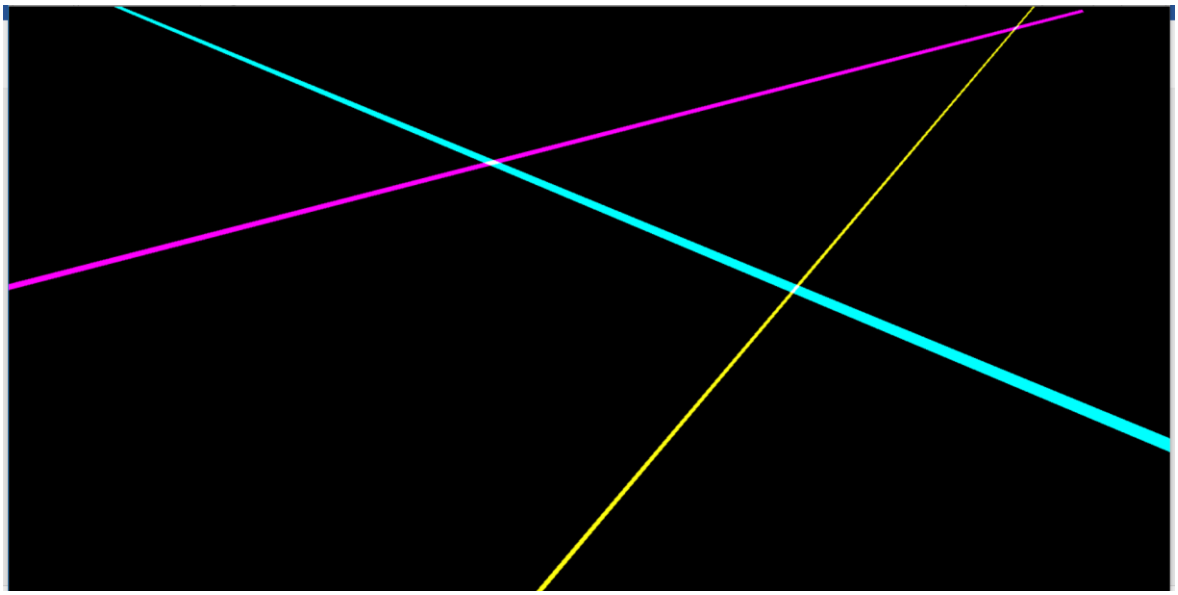

Figura 5.6 Planificador básico para espectáculo de luces 2

5.3 Resultados

Las pruebas expuestas anteriormente se han realizado a diferentes personas siguiendo el siguiente guion:

1. El tester elige un audio y se descargar en el mismo momento de realizar la prueba.
2. El testador observa los cuatro entornos de prueba y les asigna una valoración entre 0 y 10 (significando 0 que los patrones visualizados no se corresponden con los patrones rítmicos, y 10 que son perfectamente fieles al audio).
3. Se proporciona un audio común a todos a todos los testadores que realicen las pruebas, se vuelven a realizar y se las valora de nuevo.

Con ello conseguimos valoraciones de diferentes estilos de música para probar la fidelidad de la librería en diferentes escenarios, y a su vez conseguimos una valoración común al hacer que valoren un mismo audio todos.

En la siguiente tabla observaremos los audios con los que se han realizado las pruebas y sus valoraciones.

Audio	Estilo musical	PE Guitarra	PE Visualizador	PE Intensidad	PE Láser	PC Guitarra	PC Visualizador	PC Intensidad	PC Láser
Joan Manuel Serrat - Mediterraneo	Trova	10	10	10	10	10	10	10	10
Bruce Springsteen- The Rising	Rock	9	9	8	9	9	10	10	10
Klaus Badelt - Pirates of the Caribbean	Orquestal	9	9	10	10	8	9	9	10
Sia - Unstoppable	Pop	10	10	10	10	10	10	10	10
Queen - I Want to Break Free	Rock	10	10	10	10	10	10	10	10
Avicii - Waiting For Love	Electro-dance	8,5	8	8,5	9,5	8,5	9,5	9	10
Ron Gallo - Young Lady, You're Scaring	Rock	8	9	9	9	9	9	9	10
Kygo - Carry Me	Tropical house	7	8	8	9	8	10	10	10
Juanito Makandé - Niña voladora	Jazz-flamenco acústico	10	10	10	10	10	10	10	10
ACDC - Thunderstruck	Hard rock	9	10	10	10	9	9	9	9
El Fary - Apatrullando la ciudad	Copla	7,5	8,5	9	9	9	10	9	10
Alan Walker - Fade	Electrónica	9	10	9,5	10	9	10	10	10
Sebastian Ingrosso & Tommy Trash - Reload	House progresivo	8	9	9	9	9	9	9	9
Luis Fonsi ft. Daddy Yankee - Despacito	Pop latino, Reggaetón	8	9	9	10	9	9	9	10
The Who - Baba O'riley	Rock clásico	9	10	9	9	10	10	9	10
True Romance. You're so cool! - Hans Zimmer	Score, Orquestal	9	10	8	10	9	9	10	10
Queens Of The Stone Age - Go With The Flow	Hard Rock	10	8	9	9	10	9	10	10
System of a Down - Violent Pornography	Heavy metal	8	9	9	7	9	10	9,5	10
Daddy Yankee - Limbo	Electro latino	9,5	9	9	9,5	10	10	9,5	10
Ed Sheeran - Shape of You	Dance pop	9	10	9,5	10	9	10	9	10
Media	-	8,875	9,275	9,175	9,45	9,225	9,625	9,5	9,9

Tabla 5.1 Resultados de las pruebas

PE: Prueba Específica
PC: Prueba Común

La prueba común se ha realizado a partir del audio ***Safri Duo - Played-A-Live***. Esta canción destaca por su marcado ritmo de percusión, pero a su vez ofrece variaciones de intensidad y velocidad del audio, siendo una muestra que sirve para observar la finalidad de cada uno de los cuatro programas de prueba.

Como ya se ha mencionado, todas las pruebas generan los mismos eventos para un mismo audio, cambia la manera de usarlos y percibirlos visualmente, ya que cada test tiene una finalidad diferente. Al dejar libre la elección del primer audio, se han conseguido valoraciones de diferentes estilos musicales de ritmos muy diferentes, favoreciendo la fiabilidad de los resultados totales. De esta manera, las valoraciones medias oscilan satisfactoriamente entre 8,875 y 9,9, como se puede apreciar en la **Tabla 5.1**. Estos resultados muestran una fidelidad rítmica bastante exacta.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

La idea de realizar este proyecto vino a partir de otra idea más ambiciosa, pero imposible de realizar con los conocimientos adquiridos en ese momento. La idea en cuestión consistía en realizar un videojuego cuyo principal público fuesen discapacitados visuales. Para ello pretendía adaptar el deporte paralímpico Goalball, que se ha extendido poco a poco por todo el mundo, y permitir a los discapacitados disfrutarlo de una manera virtual. Para ello hacía falta conocimientos relacionados con el sonido digital, ya que sería la parte clave y fundamental en la adaptación de un deporte para ciegos. Siendo consciente de que no tenía los conocimientos necesarios para ello, decidí proponer este proyecto que unía dos de mis pasiones: Los videojuegos y la música. Esto, además me permitiría profundizar en el tema del sonido digital y adquirir nuevos conocimientos que me permitirán lograr la primera idea ya mencionada. Así, he podido aprender a trabajar con Unity 3D y conocer las entrañas del sonido digital, las cuales eran prácticamente desconocidas para mí.

Se ha conseguido afinar y mejorar la librería hasta lograr unos resultados bastante satisfactorios y agradables para los usuarios que han podido probar la librería. Estas pruebas han sido amenas y agradables para ambas partes: A mí, como desarrollador, permitiéndome mejorar poco a poco la detección de pulsos rítmicos, y pudiendo ver disfrutar a los tester escuchando sus canciones favoritas mientras veían su representación gráfica a partir del análisis de la librería. Y a los tester por poder realizar las pruebas de una manera atractiva y poco pesada. De esta manera, se ha logrado llegar a cumplir la finalidad principal de crear una librearía que analizase y generase los eventos de ritmo de un audio sin necesidad de que este mismo fuese pre-procesado o se separasen sus canales de audio.

Además, se ha podido alcanzar la finalidad secundaria de modularizar de una manera clara y detallada el código de la librería, permitiendo a futuros usuarios poder modificarla a su placer para lograr resultados diferentes, e incluso mejorarla. Es decir, es una librería que todavía permite evolución.

El punto más positivo de este proyecto, es la posibilidad de emplear la librería para desarrollar nuevas creaciones, ya sean videojuegos, aplicaciones, usos reales para espectáculos de luces, etc.

6.2 Trabajo futuro

Este trabajo podría ser el inicio de muchos proyectos interesantes. Personalmente, voy a seguir mejorando todo lo que pueda la librería y, probablemente, crear una detección melódica de los audios, lo cual abrirá más aun las puertas a la creación de nuevas aplicaciones. También empezaré un proyecto de una aplicación para simular un escenario o teatro, en el cual puedas programar sus focos, láseres y demás elementos que lo conforman, todo ello partiendo de esta librería.

De cara al usuario, al realizar este proyecto con la meta de compartirlo para cualquier interesado en usarla, será publicado de manera gratuita en la tienda de Unity, de manera que pueda servir de ayuda para nuevas creaciones, o incluso mejorar la propia librería. Me gustaría ver que los usuarios disfrutan y se divierten experimentando con la librería, poder ver sus creaciones e ideas, con el objetivo de conseguir, a través de la cooperación entre diferentes personas, llevar a cabo proyectos más grandes y ambiciosos.

Referencias

[1] John Watkinson, “The art of digital audio”, Ed.3, 2004

[2] Beat detection algorithms [Último Acceso: 30/05/2017] URL: http://archive.gamedev.net/archive/reference/programming/features/beatdetection/

[3] Luis Cavo, Siyu Tan, Adam Urga, “Beat Detection. Algorithms in Signal Processors ETIN80”, 2016-03 [Último Acceso: 30/05/2017] URL: http://www.eit.lth.se/fileadmin/eit/courses/etin80/2016/reports/beat-detection.pdf

[4] Lomont FFT library [Último Acceso: 19/03/2017] URL: http://www.lomont.org/

[5] Comunidad de Unity española [Último Acceso: 28/05/2017] URL: http://www.unityspain.com/

[6] Unity 3D main page [Último Acceso: 30/03/2017] URL: https://unity3d.com/

[7] Unity community [Último Acceso: 12/04/2017] URL: https://unity3d.com/es/learn/tutorials

[8] Unity tutorials [Último Acceso: 12/04/2017] URL: https://unity3d.com/es/learn/tutorials

[9] Occulus Rift main page [Último Acceso: 14/03/2017] URL: https://www.oculus.com/rift/

[10] GameObject Unity [Último Acceso: 30/05/2017] URL: https://docs.unity3d.com/es/current/ScriptReference/GameObject.html

[11] Mono Behaviour Unity [Último Acceso: 30/05/2017] URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

[12] Asset Store Unity [Último Acceso: 30/05/2017] URL: https://www.assetstore.unity3d.com/en/

Glosario

C#	Lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET
Unity	Unity es un motor de juego multi-desarrollado por Unity Technologies,el cual es principalmente usado para el desarrollo de videojuegos y simulaciones para PC, consolas, dispositivos móviles y sitios web.
UAM	Universidad Autónoma de Madrid.
EPS	Escuela Politécnica Superior.
FFT	Fast Fourier Transform.
IDE	Integrated Development Environment
CLR	Common Language Runtime.

Anexos

A Código relevante de la librería

- Analisis.cs

```
public Analisis(int inicio, int fin, string nombre)
{
    this.inicio = inicio;
    this.fin = fin;
    this.nombre = nombre;

    _onsets = new Dictionary<int, Onset>(3000);
    _magnitud = new List<float>();
    _flujo = new List<float>();
    _magnitudSmooth = new List<float>();
    _magnitudAvg = new List<float>();

    onsetIndices = new List<int>(3000);

    analisisDatos = new AnalisisDatos(nombre, _magnitud, _flujo, _magnitudSmooth, _magnitudAvg, _onsets);

}
```

Figura A.1 Constructor de objeto Análisis

```
public void Init(int totalFrames)
{
    this.totalFrames = totalFrames;

    onsetIndices.Clear();
    _onsets.Clear();
    _magnitud.Clear();
    _flujo.Clear();
    _magnitudSmooth.Clear();
    _magnitudAvg.Clear();

    _magnitud.Capacity = totalFrames;
    _flujo.Capacity = totalFrames;
    _magnitudSmooth.Capacity = totalFrames;
    _magnitudAvg.Capacity = totalFrames;

    _magnitud.AddRange(new float[totalFrames]);
    _flujo.AddRange(new float[totalFrames]);
    _magnitudSmooth.AddRange(new float[totalFrames]);
    _magnitudAvg.AddRange(new float[totalFrames]);

    t1 = 0;
    t2 = 0;
    p1 = 0;
    p2 = 0;

    int espectro_tam = RitmoTool.fftWindowTam / 2;
    if (fin < inicio || inicio < 0 || inicio > espectro_tam || fin > espectro_tam)
        Debug.LogError("Rango invalido para el analisis " + nombre);
}
```

Figura A.2 Inicializador de Análisis desde nuevo audio

```
public void Init(AnalisisDatos datos)
{

    onsetIndices.Clear();
    _onsets.Clear();
    _magnitud.Clear();
    _flujo.Clear();
    _magnitudSmooth.Clear();
    _magnitudAvg.Clear();

    _magnitud.AddRange(datos.magnitud);
    _flujo.AddRange(datos.flujo);
    _magnitudSmooth.AddRange(datos.magnitudSmooth);
    _magnitudAvg.AddRange(datos.magnitudAvg);

    foreach (KeyValuePair<int, Onset> onset in datos.onsets)
        _onsets.Add(onset.Key, onset.Value);
}
```

Figura A.3 Inicializador de Análisis desde datos previos

```
private void BuscaPicos(int indice, float multiplicador, int windowTam)
{
    int offset = Mathf.Max(indice - (windowTam / 2) - 1, 0);
    float th = TH(offset, multiplicador, windowTam);

    if (_flujo[offset] > th && _flujo[offset] > _flujo[offset + 1] && _flujo[offset] > _flujo[offset - 1])
    {
        Onset o = new Onset(offset, _flujo[offset], 0);
        _onsets.Add(offset, o);
        onsetIndices.Add(offset);
    }
}
```

Figura A.4 Buscador de picos en audio

```
private void RankPicos(int indice, int windowTam)
{
    int offset = Mathf.Max(0, indice - windowTam);

    if (!_onsets.ContainsKey(offset))
        return;
    int onsetId = onsetIndices.IndexOf(offset);
    int rank = onsetIndices.Count - onsetId;

    for(int i = 5; i > 0; i--)
    {
        if (onsetId - i > 0 && onsetId + i < onsetIndices.Count)
        {
            float c = _flujo[offset];
            float p = _flujo[onsetIndices[onsetId - i]];
            float n = _flujo[onsetIndices[onsetId + i]];

            if(c>p && c > n)
            {
                rank = 6 - i;
            }

            if(onsetIndices[onsetId-i]<offset-windowTam/2 && onsetIndices[onsetId+i]>offset + windowTam / 2)
            {
                rank = 6 - i;
            }
        }
    }
    _onsets[offset].rank = rank;
}
```

Figura A.5 Clasificador de picos

```
private void Smooth(int indice, int windowTam, int iteraciones)
{
    _magnitudSmooth[indice] = _magnitud[indice];

    for(int i = 1; i <= iteraciones; i++)
    {
        int iteracionId = Mathf.Max(0, indice - i * (windowTam / 2));
        Smooth(iteracionId, windowTam);
    }
}
private void Smooth(int indice, int windowTam)
{
    float avg = 0;
    for(int i = indice -(windowTam/2); i< indice + (windowTam / 2); i++)
    {
        if (i > 0 && i < totalFrames)
            avg += _magnitudSmooth[i];
    }
    _magnitudSmooth[indice] = avg / windowTam;
}
```

Figura A.6 Suavizadores de magnitud

```
private void Average(int indice)
{
    if(indice == totalFrames - 1)
    {
        t2 = indice;
        p2 = indice;
    }
    int offset = Mathf.Max(indice - 100, 1);

    if (_magnitudSmooth[offset] < _magnitudSmooth[offset - 1] && _magnitudSmooth[offset] < _magnitudSmooth[offset + 1])
    {
        t1 = t2;
        t2 = offset;
    }
    if (_magnitudSmooth[offset] > _magnitudSmooth[offset - 1] && _magnitudSmooth[offset] > _magnitudSmooth[offset + 1])
    {
        p1 = p2;
        p2 = offset;
    }
    if (t1 != t2)
    {
        if (t2 < p2)
        {
            int nt = t2 - t1;
            int np = p2 - p1;

            float ft = (_magnitudSmooth[t2] - _magnitudSmooth[t1]) / nt;
            float fp = (_magnitudSmooth[p2] - _magnitudSmooth[p1]) / np;

            for(int i =p1;i<t2; i++)
            {
                int ti = i - t1;
                int pi = i - p1;

                _magnitudAvg[i] = (_magnitudSmooth[t1] + (ft * ti)) + (_magnitudSmooth[p1] + (fp * pi));
                _magnitudAvg[i] *= .5f;
            }
            t1 = t2;
        }
    }
    if (p1 != p2)
    {
        if (p2 < t2)
        {
            int nt = t2 - t1;
            int np = p2 - p1;

            float ft = (_magnitudSmooth[t2] - _magnitudSmooth[t1]) / nt;
            float fp = (_magnitudSmooth[p2] - _magnitudSmooth[p1]) / np;

            for (int i = t1; i < p2; i++)
            {
                int ti = i - t1;
                int pi = i - p1;

                _magnitudAvg[i] = (_magnitudSmooth[t1] + (ft * ti)) + (_magnitudSmooth[p1] + (fp * pi));
                _magnitudAvg[i] *= .5f;
            }

            p1 = p2;
        }
    }
}
```

Figura A.7 Normalizador de magnitud

```
public void Analyze(float[] espectro, int indice)
{
    _magnitud[indice] = Util.Sum(espectro, inicio, fin);

    Smooth(indice, 10, 5);
    Average(indice);

    if (indice > 1)
        _flujo[indice] = (_magnitud[indice] - _magnitud[indice - 1]);

    BuscaPicos(indice, 1.9f, 12);
    RankPicos(indice - 12, 50);
}
```

Figura A.8 Función analizadora

- AnalisisDatos.cs

```
public AnalisisDatos(string nombre, List<float> magnitud, List<float>flujo,
    List<float> magnitudSmooth, List<float>magnitudAvg, Dictionary<int, Onset> onsets)
{
    this.nombre = nombre;
    this.magnitud = magnitud.AsReadOnly() ;
    this.flujo = flujo.AsReadOnly();
    this.magnitudSmooth = magnitudSmooth.AsReadOnly(); ;
    this.magnitudAvg = magnitudAvg.AsReadOnly(); ;
    this.onsets = new ReadOnlyDictionary<int, Onset>(onsets);
}
```

Figura A.9 Constructor del objeto AnalisisDatos

- Beat.cs

```
public class Beat
{
    /*Longitud del Beat*/
    public float length;

    /*bpm del Beat*/
    public float bpm;

    /*Indice del frame donde se detecta el beat*/
    public int indice;

    /*Constructor para inicializar un Beat*/
    public Beat(float length, float bpm, int indice)
    {
        this.length = length;
        this.bpm = bpm;
        this.indice = indice;
    }
}
```

Figura A.10 Constructor de objeto Beat

- BeatTracker.cs

```
public BeatTracker()
{
    _beatIndices = new List<int>(3000);
    _beats = new Dictionary<int, Beat>(3000);
    beatIndices = _beatIndices.AsReadOnly();
    beats = new ReadOnlyDictionary<int, Beat>(_beats);
}
```

Figura A.11 Constructor de objeto BeatTracker

```
public void Init(float frameLen)
{
    this.frameLen = frameLen;
    histoBeat = new float[91];
    senalBuff = new float[310];
    scoreRepe = new float[600];
    senalActual = new float[senalBuff.Length];
    senalMuestra = new float[senalActual.Length * 4];
    scoreGp = new float[180];
    scoreOffset = new float[100];

    sync = 0;
    indice = 0;
    oldSync = 0;
    mejorRepe = 0;
    lenBeatActual = 0;

    _beatIndices.Clear();
    _beats.Clear();
}
```

Figura A.12 : Inicializador de BeatTracker desde nuevo audio

```
public void Init(SongData datos)
{
    _beatIndices.Clear();
    _beatIndices.AddRange(datos.beatTracker.beatIndices);

    _beats.Clear();
    foreach (KeyValuePair<int, Beat> beat in datos.beatTracker.beats)
    {
        _beats.Add(beat.Key, beat.Value);
    }
}
```

Figura A.13 Inicializador de BeatTracker desde datos previos

```
public void TrackBeat(float muestra)
{
    senalBuff[indice % senalBuff.Length] = muestra;
    if (indice > senalActual.Length && indice % intervaloBeatTrack == 0)
    {
        for (int i = 0; i < senalActual.Length; i++)
        {
            senalActual[i] = senalBuff[(i + indice + 1) % senalBuff.Length];
        }
        oldSync = sync;
        ThreadPool.QueueUserWorkItem(o => FindBeat(), null);
    }

    if (lenBeatActual > 20)
    {
        if (sync > lenBeatActual)
        {
            sync -= lenBeatActual;
            AnnadeBeat(indice - senalActual.Length, lenBeatActual / 4f);
        }
    }
    sync += 4;

    indice++;
}
```

Figura A.14 Detector de beats

```
private void FindBeat()
{
    Util.Smooth(senalActual, 4);
    Util.Smooth(senalActual, 4);

    Array.Clear(senalActual, 0, 5);
    Array.Clear(senalActual, senalActual.Length - 5, 5);

    Util.UpsampleSingnal(senalActual, senalMuestra, 4);
    for (int i = 0; i < scoreRepe.Length; i++)
        scoreRepe[i] = ScoreRepe(senalMuestra, i);

    mejorRepe = 40;
    for (int i = 40; i < scoreRepe.Length - 5; i++)
    {
        if (scoreRepe[i] > scoreRepe[mejorRepe])
            mejorRepe = i;
    }
    float fr = scoreRepe[mejorRepe];
    for(int i = 0; i < scoreRepe.Length; i++)
    {
        scoreRepe[i] = scoreRepe[i] / fr;
    }
    for(int i = 45; i < scoreGp.Length; i++)
    {
        scoreGp[i] = ScoreGp(scoreRepe, i);
    }
    int mejorScoreGp = 45;
    for(int i = 45; i < scoreGp.Length - 1; i++)
    {
        if (scoreGp[i] > scoreGp[mejorScoreGp])
            mejorScoreGp = i;
    }

    picos.Clear();
    for(int i = 0; i < scoreRepe.Length - 5; i++)
    {
        if(i+1<scoreRepe.Length && i-1>=0)
        {
            int t = i;
            while (t < 45)
                t *= 2;
            while (t > scoreGp.Length - 1)
                t /= 2;
            if (scoreGp[t / 2] > scoreGp[t])
                t /= 2;

            float w = scoreRepe[i] * Mathf.Max(0, scoreGp[t]);

            float th = .425f;
            if (histoBeat[lenBeatActual] < 7)
                th = .25f;

            if(scoreRepe[i]>scoreRepe[i+1]&& scoreRepe[i] > scoreRepe[i - 1] && w > th)
            {
                picos.Add(i);
            }
        }
    }
```

```
    if (picos.Count == 0)
    {
        if (scoreRepe[mejorRepe / 2] > .75f)
            picos.Add(mejorRepe);
        else if (mejorRepe * 2 < scoreRepe.Length)
            if (scoreRepe[mejorRepe * 2] > .75f)
                picos.Add(mejorRepe);
    }
    if (picos.Count == 0 && lenBeatActual <= 5)
    {
        int maxPico = mejorScoreGp;
        while (maxPico < 90)
            maxPico *= 2;
        while (maxPico > 90)
            maxPico /= 2;
        picos.Add(maxPico);
    }
    picos.Insert(0, 0);
    for(int i = 0; i < picos.Count; i++)
    {
        if (i + 1 < picos.Count)
        {
            int gp = picos[i + 1] - picos[i];
            if (gp > 3)
            {
                while (gp < 45)
                {
                    gp *= 2;
                }
                while (gp > 90)
                {
                    gp /= 2;
                }
                histoBeat[gp]++;
            }
        }
    }
    lenBeatActual = 1;
    for (int i = 1; i < histoBeat.Length; i++)
    {
        if (histoBeat[i] > histoBeat[lenBeatActual])
            lenBeatActual = i;
    }
    if (histoBeat[lenBeatActual] > 15)
    {
        for (int i = 0; i < histoBeat.Length; i++)
            histoBeat[i] = Mathf.Clamp(histoBeat[i] - 7, 0, 7);
    }
    for(int i = 0; i < scoreOffset.Length; i++)
    {
        scoreOffset[i] = ScoreOffset(senalMuestra, i, lenBeatActual);
    }

    int offset = 0;
    for(int i = 1; i < scoreOffset.Length - 1; i++)
    {
        if (scoreOffset[i] > scoreOffset[offset] && scoreOffset[i] > scoreOffset[i + 1] && scoreOffset[i] > scoreOffset[i - 1])
            offset = i;
    }
    int s = 0;
    if (offset < scoreOffset.Length - 1)
    {
        offset = offset % lenBeatActual;
        s = offset;
        if (lenBeatActual > 0)
        {
            while (s < 0)
                s += lenBeatActual;
        }
        s = lenBeatActual - s;
    }
    float d = s - oldSync;
    if (Math.Abs(d) > lenBeatActual / 2f)
    {
        if (d > 0)
            sync += -2;
        else
            sync += 2;
    }
    else
    {
        if (d > 2)
            sync += 2;
        if (d < -2)
            sync += -2;
    }
}
```

Figura A.15 Actualizador de repeticiones y beats

```
public void FillInicio()
{
    if (_beatIndices.Count < 10)
        return;
    Dictionary<float, int> lens = new Dictionary<float, int>();
    int max = Mathf.Min(20, _beatIndices.Count);
    for(int i = 0; i < max; i++)
    {
        int i2 = _beatIndices[i];
        float len = _beats[i2].length;
        if (lens.ContainsKey(len))
            lens[len]++;
        else
            lens.Add(len, 1);
    }
    int nBeats = 0;
    float lenBeat = 0;
    foreach(KeyValuePair<float,int>l in lens)
    {
        if (l.Value > nBeats)
        {
            nBeats = l.Value;
            lenBeat = l.Key;
        }
    }
    if (lenBeat < 1)
        return;
    float firstId = _beatIndices[0];
    for(int i = 0; i < max; i++)
    {
        if (_beats[_beatIndices[i]].length == lenBeat)
        {
            firstId = _beatIndices[i];
            break;
        }
    }
    List<int> eliminar = new List<int>();
    foreach(KeyValuePair<int,Beat>k in _beats)
    {
        if (k.Value.indice < firstId)
            eliminar.Add(k.Key);
    }
    foreach(int k in eliminar)
    {
        _beatIndices.Remove(k);
        _beats.Remove(k);
    }
    float bpm = _beats[beatIndices[beatIndices.Count - 1]].bpm;
    while (firstId > lenBeat)
    {
        firstId -= lenBeat;
        int indice = Mathf.RoundToInt(firstId);
        _beatIndices.Insert(0, indice);
        _beats.Add(indice, new Beat(lenBeat, bpm, indice));
    }
}
```

Figura A.16 Tratamiento de beats al inicio del audio

```
public void FillFin()
{
    if (_beatIndices.Count < 40)
        return;
    Dictionary<float, int> lens = new Dictionary<float, int>();
    for(int i = beatIndices.Count - 30; i < _beatIndices.Count; i++)
    {
        int i2 = _beatIndices[i];
        float len = _beats[i2].length;
        if (lens.ContainsKey(len))
            lens[len]++;
        else
            lens.Add(len, 1);
    }
    int nBeats = 0;
    float lenBeat = 0;

    foreach(KeyValuePair<float,int>l in lens)
    {
        if (l.Value > nBeats)
        {
            nBeats = l.Value;
            lenBeat = l.Key;
        }
    }
    int lastBeat = _beatIndices[_beatIndices.Count - 1];
    int nFill = (int)((indice - lastBeat) / lenBeat);
    for (int i = 1; i < nFill; i++)
        AnnadeBeat(lastBeat + Mathf.RoundToInt(i * lenBeat), lenBeat);
}
```

Figura A.17 Tratamiento de beats al final del audio

- Segmenter.cs

```
public Segmenter(AnalisisDatos analisis)
{
    this.analisis = analisis;
    _cambios = new Dictionary<int, float>();
    _cambiosIndices = new List<int>();
    changes = new ReadOnlyDictionary<int, float>(_cambios);
    changeIndices = _cambiosIndices.AsReadOnly();
}
```

Figura A.18 Constructor de objeto Segmenter

```
public void Init(SongData datos)
{
    _cambiosIndices.Clear();
    _cambiosIndices.AddRange(datos.segmenter.changeIndices);

    _cambios.Clear();
    foreach (KeyValuePair<int, float> cambio in datos.segmenter.changes)
        _cambios.Add(cambio.Key, cambio.Value);
}
```

Figura A.19 Inicializador de Segmenter desde un audio nuevo

```
public void Init()
{
    lastDif = 0;
    aumentaIni = 0;
    aumentaFin = 0;
    aumentoDetectado = false;
    decreceIni = 0;
    decreceFin = 0;
    decrecimientoDetectado = false;

    _cambios.Clear();
    _cambiosIndices.Clear();
}
```

Figura A.20 Inicializador de Segmenter desde datos previos

```
public void DetectChanges(int indice)
{
    if (indice < 0)
        return;

    float a = analisis.magnitudAvg[indice + 1];
    float b = analisis.magnitudAvg[indice];

    float dif = a - b;

    if (dif >= .05f && lastDif < .05f && !aumentoDetectado)
    {
        aumentaIni = indice;
        aumentoDetectado = true;
    }
    if (dif <= -.08f && lastDif > -.08f && !decrecimientoDetectado)
    {
        decreceIni = indice;
        decrecimientoDetectado = true;
    }

    if (dif < .04f && lastDif > .04f && aumentoDetectado)
    {
        float rl = 22;

        if (dif > -.04f)
            rl = 12;

        aumentaFin = indice;

        aumentoDetectado = false;

        float steepest = 0;
        int si = aumentaIni;

        for (int i = aumentaIni; i < aumentaFin; i++)
        {
            float a2 = (analisis.magnitudSmooth[i + 1]);
            float b2 = (analisis.magnitudSmooth[i]);

            float nextDif = a2 - b2;

            if (nextDif > steepest)
            {
                steepest = nextDif;
                si = i;
            }
        }

        a = analisis.magnitudAvg[aumentaFin];
        b = analisis.magnitudAvg[aumentaIni];

        float length = Mathf.Sqrt(Mathf.Pow((a - b), 2) + Mathf.Pow((aumentaFin * .1f - aumentaIni * .1f), 2));

        if (length > rl)
        {
            _cambios.Add(si, a);
            _cambiosIndices.Add(si);
        }
    }
```

```
    if (dif > -.04f && lastDif < -.04f && decrecimientoDetectado)
    {
        float rl = 22;

        if (dif < .04f)
            rl = 15;

        decreceFin = indice;
        decrecimientoDetectado = false;

        float steepest = 0;
        int si = decreceIni;

        for (int i = decreceIni; i < decreceFin; i++)
        {
            float a2 = (analisis.magnitudSmooth[i + 1]);
            float b2 = (analisis.magnitudSmooth[i]);

            float nextDif = a2 - b2;

            if (nextDif < steepest)
            {
                steepest = nextDif;
                si = i;
            }
        }

        a = analisis.magnitudAvg[decreceFin];
        b = analisis.magnitudAvg[decreceIni];

        float length = Mathf.Sqrt(Mathf.Pow((a - b), 2) + Mathf.Pow((decreceFin * .1f - decreceIni * .1f), 2));

        if (length > rl)
        {
            _cambios.Add(si, -a);
            _cambiosIndices.Add(si);
        }
    }

    lastDif = dif;
}
```

Figura A.21 Detector de cambios

- EventoRitmo.cs

```
using System;
using System.Collections.Generic;
using UnityEngine;
using System.Collections.ObjectModel;
using UnityEngine.Events;

//tipo del onsetquese va a usar

public enum TipoOnset { Low,Mid,High,All }


/*
 * Componente que aporta UnityEvents. RitmoTool encontrara los EventoRitmo y los llamara
 * cuando sea necesario
 */
[AddComponentMenu("Audio/EventoRitmo")]
public class EventoRitmo: MonoBehaviour
{
    private static ReadOnlyCollection<EventoRitmo> _eventos;
    public static ReadOnlyCollection<EventoRitmo> eventos
    {
        get
        {
            if (_eventos == null)
                _eventos = eventosLista.AsReadOnly();
            return _eventos;
        }
    }
    [Tooltip("Cuantos frames se llamaran en los eventos anticipados")]
    public int targetOffset;
    [HideInInspector]
    public int offset;
```

```
    [HideInInspector]
    public float lastBeatTime;

    public int currentFrame = 0;
    public float interpolation = 0;
    public int totalFrames = 0;

    public BeatEvent onBeat = new BeatEvent();

    public SubBeatEvent onSubBeat = new SubBeatEvent();
    public OnsetEvent onOnset = new OnsetEvent();
    public ChangeEvent onChange = new ChangeEvent();
    public TimingUpdateEvent timingUpdate = new TimingUpdateEvent();

    public FrameChangedEvent onFrameChanged = new FrameChangedEvent();
    public OnNewSong onSongLoaded = new OnNewSong();
    public UnityEvent onSongEnded = new UnityEvent();
    private static List<EventoRitmo> eventosLista = new List<EventoRitmo>();
    public static event Action<EventoRitmo> EventProviderEnabled;

    void OnEnable()
    {
        if (!eventosLista.Contains(this))
        {
            eventosLista.Add(this);

            if (EventProviderEnabled != null)
                EventProviderEnabled(this);
        }
    }
    void OnDisable()
    {
        if (eventosLista.Contains(this))
            eventosLista.Remove(this);
    }
    [System.Serializable]
    public class BeatEvent : RhythmEvent<Beat>
    {
    }
    [System.Serializable]
    public class SubBeatEvent : RhythmEvent<Beat, int>
    {
    }

    [System.Serializable]
    public class TimingUpdateEvent : RhythmEvent<int, float, float, float>
    {
    }

    [System.Serializable]
    public class FrameChangedEvent : RhythmEvent<int, int>
    {
    }
    [System.Serializable]
    public class OnsetEvent : RhythmEvent<TipoOnset, Onset>
    {
    }

    [System.Serializable]
    public class ChangeEvent : RhythmEvent<int, float>
```

```
    }
    [System.Serializable]
    public abstract class RhythmEvent<T0, T1> : UnityEvent<T0, T1>
    {
        private int _listenerCount = 0;
        public int listenerCount
        {
            get
            {
                return _listenerCount + GetPersistentEventCount();
            }
        }

        new public void AddListener(UnityAction<T0, T1> call)
        {
            _listenerCount++;
            base.AddListener(call);
        }

        new public void RemoveListener(UnityAction<T0, T1> call)
        {
            _listenerCount--;
            base.RemoveListener(call);
        }
    }
    [System.Serializable]
    public abstract class RhythmEvent<T0, T1, T2, T3> : UnityEvent<T0, T1, T2, T3>
    {
        private int _listenerCount = 0;
        public int listenerCount
    {
    }

    [System.Serializable]
    public class OnNewSong : RhythmEvent<string, int>
    {
    }
    [System.Serializable]
    public abstract class RhythmEvent<T0> : UnityEvent<T0>
    {
        private int _listenerCount = 0;
        public int listenerCount
        {
            get
            {
                return _listenerCount + GetPersistentEventCount();
            }
        }

        new public void AddListener(UnityAction<T0> call)
        {
            _listenerCount++;
            base.AddListener(call);
        }

        new public void RemoveListener(UnityAction<T0> call)
        {
            _listenerCount--;
            base.RemoveListener(call);
        }
```

```
        {
            get
            {
                return _listenerCount + GetPersistentEventCount();
            }
        }

        new public void AddListener(UnityAction<T0, T1, T2, T3> call)
        {
            _listenerCount++;
            base.AddListener(call);
        }

        new public void RemoveListener(UnityAction<T0, T1, T2, T3> call)
        {
            _listenerCount--;
            base.RemoveListener(call);
        }
    }
}
```

Figura A.22 Código de la clase EventoRitmo.cs

- RitmoTool.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Threading;
using System.IO;
using UnityEngine;

[System.Serializable]
[RequireComponent(typeof(AudioSource))]
[AddComponentMenu("Audio/RitmoTool")]
public class RitmoTool : MonoBehaviour
{
    public event Action AudioLoaded;
    public event Action AudioEnded;
    public static int fftWindowTam
    {
        get
        {
            return 2048;
        }
    }
    public static int frameSpacing
    {
        get
        {
            return 1470;
        }
    }
    public ReadOnlyDictionary<int, Beat> beats
    {
        get
        {
            return beatTracker.beats;
```

```
        }
    }
    public ReadOnlyDictionary<int, float> changes
    {
        get
        {
            return segmenter.changes;
        }
    }
    public bool preCalcular
    {
        get
        {
            return _preCalcular;
        }
        set
        {
            if (analisisHecho ^ !audioLoaded)
                _preCalcular = value;
        }
    }
    public bool calcularTempo
    {
        get
        {
            return _calcularTempo;
        }
        set
        {
            if (analisisHecho ^ !audioLoaded)
                _calcularTempo = value;
        }
    }
    public bool guardarAnalisis
    {
        get
        {
            return _guardarAnalisis;
        }
        set
        {
            if (guardarAnalisis ^ !audioLoaded)
                _guardarAnalisis = value;
        }
    }
    public int lead
    {
        get
        {
            return _lead;
        }
        set
        {
            _lead = Mathf.Max(_lead, 300);
        }
    }
    public float currentSample
    {
        get;
        private set;
    }
    public int lastFrame
    {
        get;
        private set;
    }
    public int totalFrames
    {
        get;
        private set;
    }
    public int currentFrame
    {
        get;
        private set;
    }
    public float interpolation
    {
        get;
        private set;
    }
    public bool analisisHecho
    {
        get;
        private set;
    }
    public AnalisisDatos low
    {
        get
        {
            return _low.analisisDatos;
        }
```

```
    }
    public AnalisisDatos mid
    {
        get
        {
            return _mid.analisisDatos;
        }
    }
    public AnalisisDatos high
    {
        get
        {
            return _high.analisisDatos;
        }
    }
    public AnalisisDatos all
    {
        get
        {
            return _all.analisisDatos;
        }
    }
    public float bpm
    {
        get;
        private set;
    }
    public float lenBeat
    {
        get;
        private set;
    }
    public float lenFrame
    {
        get;
        private set;
    }
    public bool audioLoaded
    {
        get;
        private set;
    }
    public float volume
    {
        get
        {
            return audioSource.volume;
        }
        set
        {
            audioSource.volume = value;
        }
    }
    public float pitch
    {
        get
        {
            return audioSource.pitch;
        }
        set
        {
            audioSource.pitch = value;
        }
    }
    public bool isPlaying
    {
        get
        {
            return audioSource.isPlaying;
        }
    }
    [SerializeField]
    private bool _preCalcular = false;
    [SerializeField]
    private bool _calcularTempo = true;
    [SerializeField]
    private bool _guardarAnalisis = false;
    [SerializeField]
    private int _lead = 300;

    private BeatTracker beatTracker;
    private Segmenter segmenter;

    private Coroutine analyzeRoutine = null;
    private Coroutine queueRoutine = null;

    private float[] samples;
    private float[] monoSamples;
    private float[] spectrum;
    private int channels;

    private int lastDataFrame = 0;
    private int totalSamples;
    private int sampleIndex;
```

```
    private List<Analisis> analisis;
    private Analisis _low;
    private Analisis _mid;
    private Analisis _high;
    private Analisis _all;

    private AudioSource audioSource;

    void Init()
    {
        analisis = new List<Analisis>();

        _low = new Analisis(0, 30, "low");
        analisis.Add(_low);
        _mid = new Analisis(30, 350, "mid");
        analisis.Add(_mid);
        _high = new Analisis(370, 900, "high");
        analisis.Add(_high);
        _all = new Analisis(0, 350, "all");
        analisis.Add(_all);

        beatTracker = new BeatTracker();
        segmenter = new Segmenter(all);

        audioLoaded = false;
    }
    void Awake()
    {
        Init();
        audioSource = GetComponent<AudioSource>();
        EventoRitmo.EventProviderEnabled += OnEventProviderEnabled;
    }
    private void OnEventProviderEnabled(EventoRitmo r)
    {
        if (audioLoaded)
        {
            InitializeEventProvider(r);
        }
    }
    private void InitializeEventProvider(EventoRitmo r)
    {
        r.offset = 0;
        r.onSongLoaded.Invoke(audioSource.clip.name, totalFrames);
        r.totalFrames = totalFrames;
    }
    private void InitializeEventProviders()
    {
        foreach (EventoRitmo r in EventoRitmo.eventos)
        {
            InitializeEventProvider(r);
        }
    }
    public void NewSong(AudioClip audioClip)
    {
        if (queueRoutine != null)
            StopCoroutine(queueRoutine);

        queueRoutine = StartCoroutine(QueueNewSong(audioClip));
    }
    IEnumerator QueueNewSong(AudioClip audioClip)
    {
        yield return analyzeRoutine;

        queueRoutine = null;

        LoadNewSong(audioClip);
    }
    private void LoadNewSong(AudioClip audioClip)
    {
        audioSource.Stop();
        audioSource.clip = audioClip;

        channels = audioSource.clip.channels;
        totalSamples = audioSource.clip.samples;
        totalSamples -= totalSamples % frameSpacing;
        totalFrames = totalSamples / frameSpacing;
        lenFrame = 1 / ((float)audioSource.clip.frequency / (float)frameSpacing);

        foreach (Analisis s in analisis)
        {
            s.Init(totalFrames);
        }

        beatTracker.Init(lenFrame);
        segmenter.Init();

        samples = new float[fftWindowTam * channels];
        monoSamples = new float[fftWindowTam];
        spectrum = new float[monoSamples.Length / 2];

        currentFrame = 0;
        lastFrame = 0;
```

```
        lastDataFrame = 0;
        currentSample = 0;

        analisisHecho = false;
        audioLoaded = false;

        if (!_preCalcular)
        {
            _guardarAnalisis = false;
            analyzeRoutine = StartCoroutine(AsyncAnalyze(_lead + 300));
        }
        else
        {
            _lead = 300;
            if (_guardarAnalisis)
            {
                if (File.Exists(Application.persistentDataPath + Path.DirectorySeparatorChar + audioSource.clip.name + ".rthm"))
                {
                    SongData songData = SongData.Deserialize(audioSource.clip.name);

                    if (songData.length == totalFrames)
                    {
                        lastFrame = totalFrames;
                        analisisHecho = true;

                        foreach (AnalisisDatos data in songData.analisis)
                        {
                            foreach (Analisis a in analisis)
                            {
                                if (a.nombre == data.nombre)
                                {
                                    a.Init(data);
                                }
                            }
                        }

                        beatTracker.Init(songData);
                        segmenter.Init(songData);

                        audioLoaded = true;

                        InitializeEventProviders();

                        if (AudioLoaded != null)
                            AudioLoaded.Invoke();
                        else
                            gameObject.SendMessage("OnReadyToPlay", SendMessageOptions.DontRequireReceiver);

                        return;
                    }
                }
            }
            analyzeRoutine = StartCoroutine(AsyncAnalyze(totalFrames));
        }
    }
    private void EndOfAnalysis()
    {
        if (_calcularTempo)
            beatTracker.FillFin();

        if (_preCalcular)
        {
            if (_guardarAnalisis)
            {
                List<AnalisisDatos> datas = new List<AnalisisDatos>();

                foreach (Analisis a in analisis)
                    datas.Add(a.analisisDatos);

                SongData songData = new SongData(datas, audioSource.clip.name, totalFrames, beatTracker, segmenter);
                songData.Serialize();
            }
        }
        analisisHecho = true;
    }
    private void EndOfSong()
    {
        Stop();

        if (AudioEnded != null)
            AudioEnded.Invoke();
        else
            gameObject.SendMessage("OnEndOfSong", SendMessageOptions.DontRequireReceiver);

        for (int i = 0; i < EventoRitmo.eventos.Count; i++)
            EventoRitmo.eventos[i].onSongEnded.Invoke();
    }
    void Update()
    {
        if (!audioLoaded)
            return;

        if (audioSource.clip == null)
```

```
            return;

        if (isPlaying)
            currentSample = Mathf.Clamp(currentSample + audioSource.clip.frequency * Time.unscaledDeltaTime * audioSource.pitch,
                audioSource.timeSamples - frameSpacing / 2,
                audioSource.timeSamples + frameSpacing / 2);

        interpolation = currentSample / frameSpacing;
        currentFrame = (int)interpolation;
        interpolation -= currentFrame;

        if (currentFrame >= totalFrames - 1)
        {
            EndOfSong();
            return;
        }

        Beat nextBeat = NextBeat(currentFrame);
        lenBeat = nextBeat.length;
        bpm = nextBeat.bpm;

        Analyze();
        PassData();
    }
    private void Analyze()
    {
        if (analisisHecho)
            return;

        if (_preCalcular)
            _lead += 300;

         lead = Mathf.Max(300,  lead);
        for (int i = lastFrame + 1; i < currentFrame + _lead + 300; i++)
        {
            if (i >= totalFrames)
            {
                EndOfAnalysis();
                break;
            }


            audioSource.clip.GetData(samples, Mathf.Max((i * frameSpacing) - (samples.Length / 2), 0));

            Util.GetMono(samples, monoSamples, channels);
            Util.GetSpectrum(monoSamples);
            Util.GetSpectrumMagnitude(monoSamples, spectrum);

            foreach (Analisis s in analisis)
            {
                s.Analyze(spectrum, i);
            }

            if (_calcularTempo)
            {
                if (i - 10 >= 0)
                {
                    float flujo = all.flujo[i - 10];
                    beatTracker.TrackBeat(flujo);
                }
            }

            segmenter.DetectChanges(i - 350);

            lastFrame = i;
```

```
        }
    }
    private IEnumerator AsyncAnalyze(int frames)
    {
        frames = Mathf.Clamp(frames, 0, totalFrames);
        float[] s = new float[frames * frameSpacing * channels];
        audioSource.clip.GetData(s, 0);

        Thread analyzeThread = new Thread(BackGroundAnalyze);
        analyzeThread.Start(s);

        while (analyzeThread.IsAlive)
        {
            yield return null;
        }
        if (queueRoutine != null)
        {
            analyzeRoutine = null;
            yield break;
        }

        if (calcularTempo)
            beatTracker.FillInicio();

        if (lastFrame > totalFrames - 2)
            EndOfAnalysis();

        audioLoaded = true;

        analyzeRoutine = null;

        InitializeEventProviders();

        if (AudioLoaded != null)
            AudioLoaded.Invoke();
        else
            gameObject.SendMessage("OnReadyToPlay", SendMessageOptions.DontRequireReceiver);
    }
    private void BackGroundAnalyze(object o)
    {
        float[] s = (float[])o;

        int count = s.Length / frameSpacing / channels;

        for (int i = 0; i < count; i++)
        {
            for (int i2 = 0; i2 < samples.Length; i2++)
            {
                samples[i2] = s[Mathf.Max((i * frameSpacing * channels) - (fftWindowTam * channels), 0) + i2];
            }

            Util.GetMono(samples, monoSamples, channels);
            Util.GetSpectrum(monoSamples);
            Util.GetSpectrumMagnitude(monoSamples, spectrum);

            foreach (Analisis a in analisis)
            {
                a.Analyze(spectrum, i);
            }

            if (_calcularTempo)
            {
                if (i - 10 >= 0)
                {
                    float flujo = all.flujo[i - 10];
                    beatTracker.TrackBeat(flujo);
```

```
            }
        }

        segmenter.DetectChanges(i - 350);

        lastFrame = i;

        if (queueRoutine != null)
            break;
    }
}
private void PassData()
{
    for (int i = 0; i < EventoRitmo.eventos.Count; i++)
    {
        EventoRitmo eventoRitmo = EventoRitmo.eventos[i];

        eventoRitmo.targetOffset = Mathf.Clamp(eventoRitmo.targetOffset, 0, _lead - 25);
        eventoRitmo.currentFrame = currentFrame;
        eventoRitmo.interpolation = interpolation;

        if (eventoRitmo.offset != eventoRitmo.targetOffset)
        {
            while (eventoRitmo.offset < eventoRitmo.targetOffset)
            {
                PassSubBeat(eventoRitmo, currentFrame + eventoRitmo.offset + interpolation);
                PassEvents(eventoRitmo, currentFrame + eventoRitmo.offset);

                eventoRitmo.offset++;
            }

            eventoRitmo.offset = Mathf.Min(eventoRitmo.offset, eventoRitmo.targetOffset);
            eventoRitmo.offset = Mathf.Min(eventoRitmo.offset, eventoRitmo.targetOffset);
        }
        else
            PassSubBeat(eventoRitmo, currentFrame + eventoRitmo.offset + interpolation);

        if (eventoRitmo.timingUpdate.listenerCount != 0)
            eventoRitmo.timingUpdate.Invoke(currentFrame + eventoRitmo.offset, interpolation,
                NextBeat(currentFrame + eventoRitmo.offset).length, BeatTimer(currentFrame + eventoRitmo.offset + interpolation));
    }

    for (int i = lastDataFrame + 1; i < currentFrame + 1; i++)
    {
        for (int i2 = 0; i2 < EventoRitmo.eventos.Count; i2++)
        {
            EventoRitmo eventoRitmo = EventoRitmo.eventos[i2];

            PassEvents(eventoRitmo, i + eventoRitmo.offset);
            eventoRitmo.offset = eventoRitmo.targetOffset;
        }

        lastDataFrame = i;
    }
}
private void PassSubBeat(EventoRitmo r, float indice)
{
    float beatTime = BeatTimer(indice);
    Beat previousBeat = PreviousBeat(Mathf.CeilToInt(indice));

    if (r.lastBeatTime > beatTime)
        r.onSubBeat.Invoke(previousBeat, 0);
    if (r.lastBeatTime < .5f && beatTime >= .5f)
        r.onSubBeat.Invoke(previousBeat, 2);
    if (r.lastBeatTime < .25f && beatTime >= .25f)
        r.onSubBeat.Invoke(previousBeat, 1);
    if (r.lastBeatTime < .75f && beatTime >= .75f)
        r.onSubBeat.Invoke(previousBeat, 3);

    r.lastBeatTime = beatTime;
}
private void PassEvents(EventoRitmo r, int indice)
{
    r.onFrameChanged.Invoke(indice, lastFrame);

    if (r.onOnset.listenerCount != 0)
    {
        Onset l = _low.GetOnset(indice);
        Onset m = _mid.GetOnset(indice);
        Onset h = _high.GetOnset(indice);
        Onset a = _all.GetOnset(indice);

        if (l > 0)
            r.onOnset.Invoke(TipoOnset.Low, l);
        if (m > 0)
            r.onOnset.Invoke(TipoOnset.Mid, m);
        if (h > 0)
            r.onOnset.Invoke(TipoOnset.High, h);
        if (a > 0)
            r.onOnset.Invoke(TipoOnset.All, a);
    }

    if (r.onBeat.listenerCount != 0)
    {
        if (IsBeat(indice) == 1)
        {
            Beat beat = NextBeat(indice);
```

```
                r.onBeat.Invoke(beat);
            }
        }

        if (r.onChange.listenerCount != 0)
        {
            if (IsChange(indice))
            {
                r.onChange.Invoke(indice, NextChange(indice));
            }
        }
    }
    public void Play()
    {
        if (audioLoaded)
        {
            currentSample = audioSource.timeSamples;
            lastDataFrame = currentFrame;
            audioSource.Play();
        }
    }
    public void Stop()
    {
        if (audioLoaded)
        {
            currentSample = 0;
            audioSource.Stop();
        }
    }
    public void Pause()
    {
        audioSource.Pause();
    }
    public void UnPause()
    {
        audioSource.UnPause();
    }
    public float TimeSeconds(int indice)
    {
        return (indice * lenFrame);
    }
    public float TimeSeconds()
    {
        return TimeSeconds(currentFrame);
    }
    public void GetSpectrum(float[] samples, int channel, FFTWindow window)
    {
        audioSource.GetSpectrumData(samples, channel, window);
    }
    public AnalisisDatos AddAnalysis(int s, int e, string name)
    {
        foreach (Analisis an in analisis)
        {
            if (an.nombre == name)
            {
                Debug.LogWarning("Analysis with name " + name + " already exists");
                return null;
            }
        }

        Analisis a = new Analisis(s, e, name);
        a.Init(totalFrames);
        analisis.Add(a);
        return a.analisisDatos;
    }
    public AnalisisDatos GetAnalysis(string name)
    {
        foreach (Analisis a in analisis)
        {
            if (a.nombre == name)
                return a.analisisDatos;
        }

        Debug.LogWarning("Analysis with name " + name + " was not found");

        return null;
    }
    public void DrawDebugLines()
    {
        if (!Application.isEditor)
            return;

        if (!audioLoaded)
            return;

        for (int i = 0; i < analisis.Count; i++)
        {
            analisis[i].PintaLineasDebug(currentFrame, i);
        }

        if (_calcularTempo)
        {
            beatTracker.DrawDebugLines(currentFrame);
        }
```

```
    }
    public Beat NextBeat(int indice)
    {
        return beatTracker.NextBeat(indice);
    }

    public Beat PreviousBeat(int indice)
    {
        return beatTracker.PreviousBeat(indice);
    }
    public int NextBeatIndex(int indice)
    {
        return beatTracker.NextBeatId(indice);
    }
    public int NextBeatIndex()
    {
        return NextBeatIndex(currentFrame);
    }
    public int PrevBeatIndex()
    {
        return PrevBeatIndex(currentFrame);
    }
    public int PrevBeatIndex(int indice)
    {
        return beatTracker.PreviousBeatId(indice);
    }
    public float BeatTimer(float indice)
    {
        return beatTracker.BeatTimer(indice);
    }
    public float BeatTimer()
    {
        return BeatTimer(currentFrame + interpolation);
    }
    public int IsBeat(int indice, int min)
    {
        return beatTracker.EsBeat(indice, min);
    }
    public int IsBeat(int index)
    {
        return IsBeat(index, 0);
    }
    public bool IsChange(int index)
    {
        return segmenter.IsChange(index);
    }
    public int PrevChangeIndex(int index)
    {
        return segmenter.PrevChangeIndex(index);
    }
    public int NextChangeIndex(int index)
    {
        return segmenter.NextChangeIndex(index);
    }
    public float PrevChange(int index)
    {
        return segmenter.PrevChange(index);
    }
    public float NextChange(int index)
    {
        return segmenter.NextChange(index);
    }
}
```

Figura A.23 Código de la clase RitmoTool.cs